EBOOK

PYTHON NA PRÁTICA

APRENDA PYTHON ATRAVÉS DE EXERCÍCIOS

FERNANDO FELTRIN

Uniorg

EBOOK

PYTHON NA PRÁTICA

APRENDA PYTHON ATRAVÉS DE EXERCÍCIOS

FERNANDO FELTRIN

Uniorg

PYTHON NA PRÁTICA

EXERCÍCIOS RESOLVIDOS E COMENTADOS EM PYTHON

FERNANDO FELTRIN

Editora Uniorg

AVISOS

SOBRE O AUTOR

Fernando Feltrin é Engenheiro da Computação com especializações na área de ciência de dados e inteligência artificial, Professor licenciado para docência de nível técnico e superior, Autor de mais de 10 livros sobre programação de computadores e responsável pelo desenvolvimento e implementação de ferramentas voltadas a modelos de redes neurais artificiais aplicadas à radiologia (diagnóstico por imagem).

SISTEMÁTICA

Este livro é um tratado prático, logo, seu formato é um tanto quanto diferente do convencional por se tratar de um compêndio de exercícios resolvidos e comentados.

Crie uma lista com 8 elementos de uma lista de compras de supermercado, por meio de um laço de repetição for liste individualmente cada um dos itens dessa lista. **<= ENUNCIADO**

```
lista10 = ['Água', 'Açúcar', 'Arroz', 'Pão', 'Leite', 'Massa', 'Carne', 'Refrigerante']

for i in lista10:
  print(i)
```

<= CÓDIGO

Quando estamos a percorrer os elementos de uma lista por meio de um laço for, cada elemento dentro de seu separador será lido individualmente, tendo seu conteúdo (independentemente do tipo de dado) associado a variável temporária i. **<= EXPLICAÇÃO**

Nesse caso o retorno será:
Água
Açúcar
Arroz
Pão
Leite
Massa
Carne
Refrigerante

<= RETORNO

Sendo assim, todos os exercícios estarão dispostos no molde acima representado, facilitando o entendimento dos mesmos.

ÍNDICE

Sumário

11 - Some os valores das variáveis num1 e num2: Sendo num1 = 52 e num2 = 106. Por fim exiba em tela o resultado da soma.

12 - Some os valores das variáveis num1 e num2, atribuindo o resultado da soma a uma nova variável homônima. Exiba em tela o conteúdo dessa variável.

13 - Subtraia os valores de num1 e num2:

14 - Realize as operações de multiplicação e de divisão entre os valores das variáveis num1 e num2:

15 - Eleve o valor de num1 a oitava potência, sendo num1 = 51:

16 - Escreva um programa que pede que o usuário dê entrada em dois valores, em seguida, exiba em tela o resultado da soma, subtração, multiplicação e divisão desses números:

17 - Dadas duas variáveis num1 e num2 com valores 100 e 89, respectivamente, verifique se o valor de num1 é maior que o de num2:

18 - Verifique se os valores de num1 e de num2 são iguais:

19 - Verifique se os valores de num1 e de num2 são diferentes:

20 - Verifique se o valor de num1 é igual ou menor que 100:

21 - Verifique se os valores de num1 e de num1 são iguais ou menores que 100.

22 - Verifique se os valores de num1 ou de num2 são iguais ou maiores que 100:

23 - Verifique se o valor de num1 consta nos elementos de lista1. Sendo num1 = 100 e lista1 = [10, 100, 1000, 10000, 100000].

24 - Crie duas variáveis com dois valores numéricos inteiros digitados pelo usuário, caso o valor do primeiro número for maior que o do segundo, exiba em tela uma mensagem de acordo, caso contrário, exiba em tela uma mensagem dizendo que o primeiro valor digitado é menor que o segundo:

25 - Peça para que o usuário digite um número, em seguida exiba em tela uma mensagem dizendo se tal número é PAR ou se é ÍMPAR:

26 - Crie uma variável com valor inicial 0, enquanto o valor dessa variável for igual ou menos que 10, exiba em tela o próprio valor da variável. A cada execução a mesma deve ter seu valor atualizado, incrementado em 1 unidade.

27 - Crie uma estrutura de repetição que percorre a string 'Nikola Tesla', exibindo em tela letra por letra desse nome:

28 - Crie uma lista com 8 elementos de uma lista de compras de supermercado, por meio de um laço de repetição for liste individualmente cada um dos itens dessa lista.

29 - Crie um programa que lê um valor de início e um valor de fim, exibindo em tela a contagem dos números dentro desse intervalo.

30 - Crie um programa que realiza a contagem de 0 a 20, exibindo apenas os números pares:

31 - Crie um programa que realiza a Progressão Aritmética de 20 elementos, com primeiro termo e razão definidos pelo usuário:

32 - Crie um programa que exibe em tela a tabuada de um determinado número fornecido pelo usuário:

33 - Crie um programa que realiza a contagem regressiva de 20 segundos:

34 - Crie um programa que realiza a contagem de 1 até 100, usando apenas de números ímpares, ao final do processo exiba em tela quantos números ímpares foram encontrados nesse intervalo, assim como a soma dos mesmos:

35 - Crie um programa que pede ao usuário que o mesmo digite um número qualquer, em seguida retorne se esse número é primo ou não, caso não, retorne também quantas vezes esse número é divisível:

36 - Crie um programa que pede que o usuário digite um nome ou uma frase, verifique se esse conteúdo digitado é um palíndromo ou não, exibindo em tela esse resultado.

37 - Declare uma variável que por sua vez recebe um nome digitado pelo usuário, em seguida, exiba em tela uma mensagem

de boas vindas que incorpora o nome anteriormente digitado, fazendo uso de f'strings.

38 - Peça para que o usuário digite um número, diretamente dentro da função print( ) eleve esse número ao quadrado, exibindo o resultado incorporado a uma mensagem:

39 - Dada a seguinte lista: nomes = ['Ana', 'Carlos', 'Daiane', 'Fernando', 'Maria'], substitua o terceiro elemento da lista por 'Jamile':

40 - Adicione o elemento 'Paulo' na lista nomes:

41 - Adicione o elemento 'Eliana' na lista nomes, especificamente na terceira posição da lista:

42 - Remova o elemento 'Carlos' da lista nomes:

43 - Mostre o segundo, terceiro e quarto elemento da lista nomes. Separadamente, mostre apenas o último elemento da lista nomes:

44 – Crie um dicionário via método construtor dict( ), atribuindo para o mesmo ao menos 5 conjuntos de chaves e valores representando objetos e seus respectivos preços:

45 – Crie um dicionário usando o construtor de dicionários do Python, alimente os valores do mesmo com os dados de duas listas:

46 - Crie uma simples estrutura de dados simulando um cadastro para uma loja. Nesse cadastro deve conter informações como nome, idade, sexo, estado civil, nacionalidade, faixa de renda, etc... Exiba em tela tais dados:

47 - Crie um programa que recebe dados de um aluno como nome e suas notas em supostos 3 trimestres de aula, retornando um novo dicionário com o nome do aluno e a média de suas notas:

48 - Crie um sistema de perguntas e respostas que interage com o usuário, pedindo que o mesmo insira uma resposta. Caso a primeira questão esteja correta, exiba em tela uma mensagem de acerto e parta para a próxima pergunta, caso incorreta, exiba uma mensagem de erro e pule para próxima pergunta:

49 - Crie uma função de nome funcao1, que por sua vez não realiza nenhuma ação:

50 - Atribua a função funcao1 a uma variável:

51 - Crie uma função que retorna um valor padrão:

52 - Crie uma função que exibe em tela uma mensagem de boas-vindas:

53 - Crie uma função que recebe um nome como parâmetro e exibe em tela uma mensagem de boas-vindas. O nome deve ser fornecido pelo usuário, incorporado na mensagem de boas-vindas da função:

54 - Crie uma função que recebe um valor digitado pelo usuário e eleva esse valor ao quadrado:

55 - Crie uma função com dois parâmetros relacionados ao nome e sobrenome de uma pessoa, a função deve retornar uma mensagem de boas-vindas e esses dados devem ser digitados pelo usuário:

56 - Crie uma função com dois parâmetros, sendo um deles com um dado/valor predeterminado:

57 - Crie uma função com três parâmetros, sendo dois deles com dados/valores padrão, alterando o terceiro deles contornando o paradigma da justaposição de argumentos:

58 - Crie uma função que pode conter dois ou mais parâmetros, porém sem um número definido e declarado de parâmetros:

59 - Crie uma função de número de parâmetros indefinido, que realiza a soma dos números repassados como parâmetro, independentemente da quantidade de números:

60 - Crie uma função que recebe parâmetros tanto por justaposição (*args) quanto nomeados (**kwargs):

61 - Escreva um programa que retorna o número de Fibonacci: Sendo o número de Fibonacci um valor iniciado em 0 ou em 1 onde cada termo subsequente corresponde à soma dos dois anteriores.

62 - Crie um programa modularizado, onde em um arquivo teremos uma lista de médicos fictícios a serem consultados, em outro arquivo, teremos a estrutura principal do programa, que por sua vez realiza o agendamento de uma consulta médica com base na interação com o usuário.

63 - Aprimore o exemplo anterior, incluindo um módulo simulando o cadastro de usuários em um plano de saúde, apenas permitindo o agendamento de consulta caso o usuário que está interagindo com o programa conste no cadastro:

64 - Crie uma função que recebe parâmetros tanto por justaposição quanto nomeados a partir de uma lista e de um dicionário, desempacotando os elementos e reorganizando os mesmos como parâmetros da função:

65 - Crie uma classe de nome Carro e lhe dê três atributos: nome, ano e cor.

66 - Crie uma classe Pessoa, instancie a mesma por meio de uma variável e crie alguns atributos de classe dando características a essa pessoa. Por fim exiba em tela alguma mensagem que incorpore os atributos de classe criados:

67 - Crie uma classe que armazena algumas características de um carro, em seguida crie dois carros distintos, de características diferentes, usando da classe para construção de seus objetos/variáveis.

68 - Crie uma classe Pessoa com método inicializador e alguns objetos de classe vazios dentro da mesma que representem características de uma pessoa:

69 - Crie uma classe Inventario com os atributos de classe pré-definidos item1 e item2, a serem cadastrados manualmente pelo usuário, simulando um simples carrinho de compras:

70 - Crie uma classe Biblioteca que possui uma estrutura molde básica para cadastro de um livro de acordo com seu título, porém que espera a inclusão de um número não definido de títulos. Em seguida cadastre ao menos 5 livros nessa biblioteca:

71 - Crie uma calculadora simples de 4 operações (soma, subtração, multiplicação e divisão) usando apenas de estrutura de código orientada a objetos:

72 - Mostre via terminal a string ‘Bem vindo ao mundo da programação!!!’ de trás para frente usando indexação:

73 - Escreva um programa que encontre todos os números que são divisíveis por 7, mas que não são múltiplos de 5, entre 2000 e 2200 (ambos incluídos). Os números obtidos devem ser impressos em sequência, separados por vírgula e em uma única linha:

74 - Escreva um programa, uma calculadora simples de 4 operações, onde o usuário escolherá entre uma das 4 operações (soma, subtração, multiplicação e divisão). Lembrando que o usuário digitará apenas dois valores, e escolherá apenas uma operação matemática do menu.

75 - Crie uma função que recebe um número e retorna o mesmo dividido em duas metades, sendo cada metade um elemento de uma lista:

76 - Crie um programa que gera um dicionário a partir de um valor digitado pelo usuário, de modo que serão exibidos todos valores antecessores a este número multiplicados por eles mesmos. Supondo que o usuário tenha digitado 4, a saída deve ser {1: 1, 2: 4, 3: 9, 4: 16} :

77 - Defina uma função que pode aceitar duas strings como entrada, exibindo em tela apenas a string de maior tamanho/comprimeito. Caso as duas strings tenham mesmo tamanho, exiba em tela as duas:

78 - Escreva um programa que recebe um texto do usuário e o converte para código Morse, exibindo em tela o texto em formato Morse, seguindo a padronização “ . - " (ponto, traço).

79 - Escreva um programa que recebe do usuário um número de 0 a 100 e transcreve o mesmo por extenso. Por exemplo o usuário digita 49, então é retornara a string ‘quarenta e nove’.

80 - Crie uma função que recebe um nome e um sobrenome do usuário, retornando os mesmos no padrão americano, ou seja, sobrenome primeiro, seguido do nome:

81 - Crie um programa que gera uma senha aleatória, com um tamanho definido pelo usuário:

82 - Crie uma função que exibe em tela tanto o conteúdo de uma variável local quanto de uma variável global, sendo as variáveis de mesmo nome, porém uma não substituindo a outra:

83 – Crie um programa que recebe um número digitado pelo usuário, convertendo o mesmo para algarismo de número Romano, exibindo em tela esse dado já convertido:

84 – Crie um programa que mostra a data atual, formatada para dia-mês-ano, hora:minuto:segundo.

85 – Crie um programa que exibe a versão atual do Python instalada no sistema:

86 – A partir de uma lista composta apenas de dados numéricos, gere outra lista usando de list comprehension usando como base a lista anterior, compondo a nova com os valores dos elementos originais elevados ao cubo:

87 – Dada uma estrutura inicial de um carrinho de compras com 5 itens e seus respectivos valores, assim como uma função que soma os valores dos itens do carrinho, retornando um valor total. Aprimore a função deste código usando de list comprehension:

88 – Escreva uma função que recebe do usuário um número qualquer e retorna para o mesmo tal número elevado ao quadrado. Crie uma documentação para essa função que possa ser acessada pelo usuário diretamente via IDE.

89 – Escreva um programa que recebe uma frase qualquer, mapeando a mesma e retornando ao usuário cada palavra com a frequência com que a mesma aparece na frase em questão.

90 – Crie um programa que recebe do usuário uma sequência de números aleatórios separados por vírgula, armazene os números um a um, em formato de texto, como elementos ordenados de

uma lista. Mostre em tela a lista com seus respectivos elementos após ordenados.

91 – Escreva um programa, da forma mais reduzida o possível, que recebe do usuário uma série de nomes, separando os mesmos e os organizando em ordem alfabética. Em seguida, exiba em tela os nomes já ordenados.

92 – Escreva um simples programa que recebe do usuário um número qualquer, retornando ao mesmo se este número digitado é um número perfeito.

93 – Escreva uma função que recebe uma lista de elementos totalmente aleatórios e os ordena de forma crescente de acordo com seu valor numérico:

94 – Crie uma estrutura toda orientada a objetos que recebe do usuário uma string qualquer, retornando a mesma com todas suas letras convertidas para letra maiúscula. Os métodos de classe para cada funcionalidade devem ser independentes entre si, porém trabalhar no escopo geral da classe.

95 – Escreva de forma reduzida um programa que recebe do usuário um nome e duas notas, salvando tais dados como um elemento de uma lista. Exiba em tela o conteúdo dessa lista.

96 – Crie um programa que gera o valor de salário de funcionários considerando apenas horas trabalhadas e horas extras, sendo o valor fixo da hora trabalhada R$29,11 e da hora extra R$5,00. Crie uma regra onde o funcionário só tem direito a receber horas extras a partir de 40 horas trabalhadas da forma convencional.

97 – Reescreva o código anterior adicionando um mecanismo simples de validação que verifica se os dados inseridos pelo usuário são de tipos numéricos, caso não sejam, que o processo seja encerrado.

98 – Crie um programa que recebe uma nota entre 0 e 1.0, classificando de acordo com a nota se um aluno fictício está aprovado ou em recuperação de acordo com sua nota. A média para aprovação deve ser 0.6 ou mais, e o programa deve realizar

as devidas validações para caso o usuário digite a nota em um formato inválido.

99 – Crie uma estrutura molde (orientada a objetos) para cadastro de veículos, tendo como características que os descrevem sua marca, modelo, ano, cor e valor. Cadastre ao menos três veículos, revelando seu número identificador de objeto alocado em memória, assim como o retorno esperado pelo usuário quando o mesmo consultar tal veículo.

100 – Crie um programa que recebe do usuário três números diferentes, retornando para o mesmo qual destes números é o de maior valor. Usar de funções aninhadas para realizar as devidas comparações entre os valores dos números:

101 – Crie um programa que atua como mecanismo controlador de um sistema direcional, registrando a direção e o número de passos executado. Ao final do processo, exiba em tela o número de referência para um plano cartesiano. Ex: Se o usuário digitar CIMA 5, BAIXO 3, ESQUERDA 3 e DIREITA 2, o resultado será a coordenada 2.

102 – Crie uma estrutura orientada a objetos híbrida, para cadastro de usuários, que permita ser instanciada por uma variável de modo que a mesma pode ou não repassar argumentos para seus objetos de classe na hora em que instancia tal classe. Para isso, a classe deve conter atributos/objetos de classe padrão e instanciáveis de fora da classe.

103 – Gere uma lista com 50 elementos ordenados de forma crescente. Inicialmente usando do método convencional (e do construtor de listas padrão), posteriormente reescrevendo o código usando de list comprehension.

104 – Reescreva a estrutura de código abaixo, de um simples programa que gera uma lista com as possíveis combinações entre duas outras listas [1,2,3] e [4,5,6], reduzindo e otimizando a mesma via list comprehension:

105 – Escreva um programa que cria uma array de 5 elementos do tipo int. Exiba em tela todos os elementos da array, em seguida

exiba individualmente apenas os elementos pares da mesma de acordo com sua posição de índice.

106 – Crie uma array composta de 6 elementos aleatórios, porém com valores sequenciais crescentes. Uma vez criada a array, exiba em tela seu conteúdo em seu formato original, e em seu formato invertido.

107 – Dada a seguinte lista simples de elementos [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21], transforme a mesma em um objeto do tipo array, em seguida exiba em tela o tamanho dessa array (número de elementos), separadamente, exiba o tamanho da array em bytes e seu marcador de posição alocada em memória.

108 – Escreva um programa que, usando puramente de lógica, encontra todos os números que são divisíveis por 7 e múltiplos de 5, dentro do intervalo de 0 a 500 (incluindo tais valores).

109 – Escreva um programa que gera um número aleatório entre 0 a 10, salvando esse número em uma variável, onde o usuário é convidado a interagir tentando adivinhar qual foi o número gerado até acertar. Caso o usuário acerte o número, exiba uma mensagem o parabenizando, também exibindo incorporado em uma mensagem o número em si.

110 – Escreva um programa que executa um bloco de código situado dentro de um comentário.

111 – Escreva um programa que recebe uma palavra qualquer do usuário, aplicando sobre a mesma uma formatação de estilo, retornando a mesma com marcadores para negrito, itálico e sublinhado. As funções que aplicam os estilos devem ser separadas entre si, porém executadas em conjunto via decoradores para uma função principal.

112 – Escreva um programa que lê uma palavra ou frase digitada pelo usuário, retornando o número de letras maiúsculas e minúsculas da mesma. Usar apenas de lógica e de funções embutidas ao sistema.

113 – Escreva um programa que verifica se uma determinada palavra consta em um texto de origem. Texto: “Python é uma

excelente linguagem de programação!!!”.

114 – Escreva um programa que verifica se um endereço de site foi digitado corretamente, validando se em tal endereço constam o prefixo ‘www.’ E o sufixo ‘.com.br’, pedindo que o usuário digite repetidas vezes o endereço até o mesmo for validado como correto.

115 – Escreva um programa que recebe uma lista de elementos mistos, trocando de posições apenas o primeiro e o último elemento. Lista de referência [11, 22, 33, 44, 55, 66, 77, 88, 99, ‘X’].

117 – Escreva um programa que realiza a contagem do número de segundas feiras que caem no dia 1º de cada mês, dentro do intervalo do ano 2010 até o ano de 2020.

118 – Escreva um programa que exibe uma mensagem em formato de contagem regressiva, contando de 10 até 0, com o intervalo de 2 segundos entre uma mensagem e outra.

119 – Escreva um programa que retorna a data e hora da última vez em que um determinado arquivo foi modificado.

120 – Escreva um programa que extrai os dados / valores dos elementos de uma tupla, que representa uma escala de dias de atendimento, para suas respectivas variáveis individuais. Exiba em tela uma mensagem referente aos dias 3 e 4 da escala.

121– Crie uma função reduzida para separar elementos negativos e positivos de uma lista, sem aplicar qualquer tipo de ordenação sobre os mesmos.

122 – Escreva um programa que retorna os 5 elementos mais comuns de um texto (caracteres individuais), exibindo em tela tanto os elementos em si quanto o número de incidências de tais elementos.

123 – Escreva um programa que recebe três conjuntos numéricos de mesmos valores, o terceiro conjunto deve ser uma cópia literal do segundo. Exiba em tela os conteúdos dos conjuntos, seu identificador de memória e se alguns destes conjuntos

INTRODUÇÃO

Se tratando de programação, independentemente da linguagem, um dos maiores erros dos estudantes ou entusiastas da área é o fato de os mesmos apenas se limitarem a replicar os exemplos dos conceitos da linguagem, não praticando de fato, seja explorando outras possibilidades de mesmo propósito, seja não resolvendo exercícios que estimulem a aplicação da lógica e implementação das ferramentas que a linguagem de programação oferece para resolução de problemas computacionais.

Sendo assim, o intuito deste pequeno livro é trazer à tona desde os conceitos mais básicos da linguagem Python até tópicos específicos e avançados por meio de exercícios, de modo que cada conceito será revisado conforme a necessidade de sua aplicação, assim como para programadores que já possuem uma bagagem de conhecimento, tais exemplos servirão para os mesmos realizar engenharia reversa nos conceitos entendendo os porquês de cada estrutura de código.

Esse livro foi escrito de forma que os exercícios estão dispostos de forma gradual no que diz respeito à sua dificuldade, todo e qualquer exercício terá suas devidas explicações para cada linha de código, porém de forma gradual à medida que avançaremos pelos exercícios iremos cada vez nos atendo mais às suas particularidades do que conceitos iniciais. Em função disso, faça os exercícios em sua ordem, não avançando quando encontrada alguma dificuldade, assim como quando se sentir confiável, tente outros métodos de resolução dos mesmos exercícios.

Dessa forma, garanto que você aprenderá de uma vez por todas como aplicar seus conhecimentos em problemas computacionais reais.

Então... vamos direto para prática!

EXERCÍCIOS

1 - Crie três variáveis com três tipos de dados diferentes, respeitando sua sintaxe:

```
nome = 'Fernando'
ano = 1987
valor = 19.99
```

Para declarar uma variável basta dar um nome para a mesma, seguido do operador de atribuição " = " e do conteúdo a ser atribuído para essa variável.

A forma como o atributo da variável é escrita fará com que o interpretador identifique automaticamente o tipo de dado em questão. Nesse caso, para a variável nome temos uma string, da mesma forma para variável ano temos um int assim como para variável valor temos um dado atribuído do tipo float.

---

2 - Crie um comentário de no máximo uma linha:

```
# Exemplo de comentário simples, de até uma linha.
```

Comentários curtos em Python devem iniciar com o símbolo “ # “, dessa forma o conteúdo desta linha de código será simplesmente ignorado pelo interpretador.

---

3 - Crie um comentário de mais de uma linha:

```
'''Apenas explorando a possibilidade de criar um
  comentário composto de mais de uma linha, que
  permite uma explicação mais elaborada sobre o
  código em questão.'''
```

Todo comentário maior que uma linha em Python deve ter seu conteúdo entre aspas triplas simples ‘’’ ‘’’ ou aspas triplas duplas “”” “””, de acordo com a preferência do desenvolvedor.

---

4 - Escreva um programa que mostra em tela a mensagem: Olá Mundo!!!

```
print('Olá Mundo!!!')
```

Usando da função print( ) por sua vez parametrizada com a string ‘Olá Mundo!!!’, ao executar o código será exibido em tela / em terminal a mensagem Olá Mundo!!!.

O retorno será:
Olá Mundo!!!

---

5 - Crie uma variável nome e atribua para a mesma um nome digitado pelo usuário:

```
nome = input('Por favor, digite o seu nome: ')
```

De acordo com o enunciado do exercício, inicialmente declaramos uma variável de nome nome, que por meio da função input( ) pede que o usuário digite algo, nesse caso, seu próprio nome.

Ao terminar de digitar o nome e pressionar a tecla ENTER, o conteúdo digitado estará atribuído para a variável nome.

---

6 - Exiba em tela o valor e o tipo de dado da variável num1: Sendo num1 = 1987.

```
num1 = 1987

print(num1)
print(type(num1))
```

Inicialmente é declarada uma variável de nome num1, que por sua vez tem atribuída o número int 1987. Parametrizando a função print( ) com a variável num1, obteremos como retorno o valor da mesma.

Já quando parametrizamos nossa função print( ) com a função type( ) por sua vez parametrizada com a variável num1, o retorno esperado é que seja exibido em tela o tipo de dado associado a essa variável.

O retorno será:
1987
Int

---

7 - Peça para que o usuário digite um número, em seguida exiba em tela o número digitado.

```
num = input('Digite um número: ')

print(f'O número digitado é : {num}')
```

Declarada a variável num, por sua vez parametrizada com a função input( ) que interage com o usuário pedindo que o mesmo digite um número, podemos exibir esse número em tela via função print( ).

Usando de f'strings e máscaras de substituição podemos criar um retorno um pouco mais elaborado, incorporando na string o conteúdo da variável num.

Supondo que o usuário tenha digitado 106, o retorno será:
O número digitado é: 106

---

8 - Peça para que o usuário digite um número, em seguida o converta para float, exibindo em tala tanto o número em si quanto seu tipo de dado.

```
num = input('Digite um número: ')
num = float(num)

print(num)
print(type(num))
```

Uma vez criada nossa variável num, com seu conteúdo vindo da interação com o usuário, podemos atualizar o conteúdo dessa variável, mudando inclusive seu tipo de dado. Para isso, nossa variável num recebe como atributo o método float( ) parametrizado com ela mesma.

Dessa forma, o conteúdo atribuído a variável num é convertido de formato e salvo sobrescrevendo o conteúdo antigo dessa variável. Por meio da função print( ) podemos exibir em tela tanto o tipo quanto o conteúdo da variável num.

Supondo que o usuário tenha digitado 52, o retorno será:
52
float

---

9 - Crie uma lista com 5 nomes de pessoas:

```
nomes = ['Ana', 'Carlos', 'Daiane', 'Fernando', 'Maria']

print(nomes)
```

Uma lista em Python, tipo de dado muito similar a uma array em outras linguagens de programação, é uma estrutura de dados que serve para guardar mais de um dado associado a uma variável, sem distinção da quantia nem do tipo de dado em questão, a única ressalva é que para que tal tipo de estrutura de dado seja reconhecida corretamente pelo interpretador do Python, a mesma deve obedecer a sintaxe, toda e qualquer lista tem seus dados dentro de " [ ] " colchetes.

O retorno será:
[‘Ana’, ‘Carlos’, ‘Daiane’, ‘Fernando’, ‘Maria’]

---

10 - Mostre o tamanho da lista nomes / o número de elementos da lista nomes:

Mostre separadamente apenas o terceiro elemento dessa lista:

```
nomes = ['Ana', 'Carlos', 'Daiane', 'Fernando', 'Maria']

print(len(nomes))

print(nomes[3])
```

Em nossa função print( ) usando do método len( ) por sua vez parametrizado com a variável nomes, o retorno esperado é o número de elementos que compõe essa lista.

Para instanciar apenas um determinado elemento, devemos fazer referência ao mesmo por meio de seu número de índice. Cada elemento de uma lista possui internamente um valor de índice, sendo o primeiro elemento o valor 0, o segundo elemento o valor 1, e assim por diante.

Para exibir em tela apenas o terceiro elemento da lista nomes, basta parametrizar nossa função print( ) com nomes[ ] fazendo referência ao índice de número 3.

O retorno será:
5
Fernando

---

11 - Some os valores das variáveis num1 e num2: Sendo num1 = 52 e num2 = 106. Por fim exiba em tela o resultado da soma.

```
num1 = 52
num2 = 106

print(num1 + num2)
```

Declaradas as variáveis num1 e num2 com seus respectivos atributos, podemos diretamente dentro da função print( ) executar a operação de soma destas variáveis, bastando instanciar as mesmas em nossa função print e definindo a operação de soma por meio do operador " + ".

O retorno será:
158

---

12 - Some os valores das variáveis num1 e num2, atribuindo o resultado da soma a uma nova variável homônima. Exiba em tela o conteúdo dessa variável.

```
num1 = 52
num2 = 106

soma = num1 + num2

print(soma)
```

No exemplo anterior, ao executar a operação matemática diretamente como parâmetro de nossa função print( ) o resultado, foi exibido em tela porém após o encerramento desse bloco de código tal dado, como esperado, foi descarregado da memória.

Para guardar essa informação de forma "permanente" é necessário realizar a operação atribuída a uma variável. Desse modo, podemos exibir o dado em tela parametrizando a função print( ) com a variável em questão, e encerrado o processo o dado continua guardado na variável para reutilização.

O retorno será:
158

---

13 - Subtraia os valores de num1 e num2:

```
num1 = 52
num2 = 106

print(num1 - num2)

# ou

subtracao = num1 - num2
print(subtracao)
```

Para subtrair o valor atribuído a num1 do valor atribuído a num2, basta fazer o uso do operador " – ", diretamente em nossa função print( ) ou atribuindo esse resultado a uma variável, como já vimos anteriormente.

O retorno será:
-54

---

14 - Realize as operações de multiplicação e de divisão entre os valores das variáveis num1 e num2:

```
num1 = 52
num2 = 106

print(num1 * num2)

print(num1 / num2)
```

*Usando das mesmas variáveis com seus respectivos valores, como nos exemplos anteriores, para multiplicarmos os valores dessas variáveis basta usar do operador " * ", da mesma forma, para realizar a divisão simples dos valores de num1 e num2, usamos do operador " / ", obtendo assim o resto da divisão.*

---

15 - Eleve o valor de num1 a oitava potência, sendo num1 = 51:

```
num1 = 51

print(num1 ** 8)
```

*Para efetuar uma exponenciação em Python simplesmente usamos do operador “ ** “.*

O retorno de 51 elevado à oitava potência, nesse caso é o valor que equivale a 51 x 51 x 51 x 51 x 51 x 51 x 51 x 51, resultando em 45767944570401.

---

16 - Escreva um programa que pede que o usuário dê entrada em dois valores, em seguida, exiba em tela o resultado da soma, subtração, multiplicação e divisão desses números:

```
num1 = int(input('Digite o primeiro número: '))
num2 = int(input('Digite o segundo número: '))

print(f'A soma de {num1} com {num2} é: {num1 + num2}')
print(f'A subtração de {num1} com {num2} é: {num1 - num2}')
print(f'A multiplicação de {num1} com {num2} é: {num1 * num2}')
print(f'A divisão de {num1} por {num2} é: {num1 / num2}')
```

Como feito em exemplos anteriores, por meio da função input( ) podemos pedir que o usuário digite dois números que ficarão atribuídos a duas variáveis distintas num1 e num2.

Em seguida, por meio de nossa função print( ) por sua vez parametrizada por uma f'string, conseguimos em máscaras de substituição instanciar tanto os valores de num1 quanto os valores de num2, realizando também as devidas operações matemáticas dentro de uma máscara de substituição.

---

17 - Dadas duas variáveis num1 e num2 com valores 100 e 89, respectivamente, verifique se o valor de num1 é maior que o de num2:

```
num1 = 100
num2 = 89

print(num1 > num2)
```

Nesse caso, declaradas as variáveis com seus respectivos valores podemos diretamente via função print( ) realizar a comparação lógica. Para isso, simplesmente parametrizamos nossa função print( ) com as variáveis em questão, usando entre as mesmas o operador lógico " > " para verificar se o valor atribuído a num1 é maior que o valor atribuído a num2.

Nesse caso, por se tratar de uma expressão lógica, o retorno será: True.

---

18 - Verifique se os valores de num1 e de num2 são iguais:

```
num1 = 100
num2 = 89

print(num1 == num2)
```

Para verificar de forma lógica se os valores de duas variáveis é igual ou ao menos equivalente, usamos do operador lógico " == ", que para esse exemplo retornará False, uma vez que os valores das variáveis em questão de fato não são iguais.

---

19 - Verifique se os valores de num1 e de num2 são diferentes:

```
num1 = 100
num2 = 89

print(num1 != num2)
```

De forma parecida com o exemplo anterior, agora usando do operador lógico " != " podemos verificar se os conteúdos de duas variáveis são diferentes. Nesse caso, o esperado é que seja retornado True, pois os valores atribuídos para num1 e num2 são diferentes entre si.

---

20 - Verifique se o valor de num1 é igual ou menor que 100:

```
num1 = 100

print(num1 <= 100)
```

Lembrando que aqui temos uma estrutura condicional composta, ou igual ou menor que, bastando apenas uma dessas proposições ser verdadeira para que o retorno gerado seja True. Em nosso exemplo, num1 não é menor mas é igual a 100, logo, o retorno será True.

---

21 - Verifique se os valores de num1 e de num1 são iguais ou menores que 100.

```
num1 = 100
num2 = 89

print(num1 <= 100 and num2 <= 100)
```

Para esse exemplo temos de fato uma estrutura condicional composta, onde além das condições de cada expressão, temos o operador and que determina que ambas expressões também sejam verdadeiras para que seja obtido um retorno True.

Em outras palavras temos na primeira expressão "o valor de num1 é menor ou é igual a 100" assim como na segunda expressão temos "o valor de num2 é menor ou é igual a 100", entre as mesmas temos o operador and, que por sua vez coloca a condição "a primeira expressão e a segunda expressão são verdadeiras?", somente retornando True caso ambas sejam.

Pela tabela verdade AND, True e True = True. (caso uma das expressões fosse False, o retorno seria False. Ex: True e False = False, False e True = False.)

**A mesma expressão poderia ser reescrita de forma reduzida como: print(num1 and num2 <= 100), gerando o mesmo retorno.*

---

22 - Verifique se os valores de num1 ou de num2 são iguais ou maiores que 100:

```
num1 = 100
num2 = 89

print(num1 >= 100 or num2 >= 100)
```

Diferentemente do exercício anterior, agora usando do operador or podemos criar expressões lógicas onde basta uma das expressões ser verdadeira para já validar todo o contexto.

Note que na primeira expressão, o valor de num1 não é maior, mas é igual a 100, já na segunda expressão o valor de num2 não é maior, mas é igual a 100, em função do operador or, mesmo caso uma dessas expressões fosse False o retorno seria True.

---

23 - Verifique se o valor de num1 consta nos elementos de lista1. Sendo num1 = 100 e lista1 = [10, 100, 1000, 10000, 100000].

```
num1 = 100

lista1 = [10, 100, 1000, 10000, 100000]

print(num1 in lista1)
```

Aqui temos uma expressão lógica fazendo o uso do operador in, que basicamente nos é útil para verificar se um determinado dado/valor consta dentro de uma variável/objeto.

Nesse caso, se o valor atribuído para num1 constar como um dos elementos de lista1, o retorno será True.

---

24 - Crie duas variáveis com dois valores numéricos inteiros digitados pelo usuário, caso o valor do primeiro número for maior que o do segundo, exiba em tela uma mensagem de acordo, caso contrário, exiba em tela uma mensagem dizendo que o primeiro valor digitado é menor que o segundo:

```
num1 = int(input('Digite o primeiro número: '))
num2 = int(input('Digite o segundo número: '))

if num1 > num2:
  print('O primeiro número digitado é o maior!')
else:
  print('O segundo número digitado é o maior!')
```

Lembrando que em uma estrutura condicional simples, criamos um objetivo a ser alcançado/atingido, indentando blocos de código de acordo com as condições importas.

Nesse caso, como temos apenas dois possíveis desfechos de acordo com a condição, supondo que o usuário digitou 25 e 26, respectivamente, o resultado exibido em tela seria 'O segundo número digitado é o maior', uma vez que a primeira condição (se num1 for maior que num2) não é válida.

---

25 - Peça para que o usuário digite um número, em seguida exiba em tela uma mensagem dizendo se tal número é PAR ou se é ÍMPAR:

```
num = int(input('Digite um número: '))

if (num % 2) == 0:
    print(f'{num} é PAR')
else:
    print(f'{num} é ÍMPAR')
```

Após criar a linha de código responsável por pedir ao usuário que o mesmo digite um número, validando esse número como do tipo inteiro e atribuindo o número em si a uma variável, vamos a estrutura condicional.

Para verificar se um determinado número é par, simplesmente o resto da divisão desse número por 2 deve ser igual a 0. Logo, criamos uma condição onde se o resto da divisão do valor de num por 2 for igual a 0, é exibida uma mensagem dizendo que o mesmo é PAR, caso essa condição não seja verdadeira, é exibida em tela uma outra mensagem, dessa vez dizendo que o número em questão é ÍMPAR.

Supondo que o usuário tenha digitado o número 15, o retorno gerado é ’15 é ÍMPAR’.

---

26 - Crie uma variável com valor inicial 0, enquanto o valor dessa variável for igual ou menos que 10, exiba em tela o próprio valor da variável. A cada execução a mesma deve ter seu valor atualizado, incrementado em 1 unidade.

```
var = 0

while var <= 10:
  print(var)
  var = var + 1
```

Toda vez que se faz necessário repetir uma determinada instrução enquanto um determinado objetivo não é atingido, podemos realizar essa repetição por meio da estrutura de repetição while.

Quando usamos while, definimos uma condição a ser alcançada ou validada, e para que a execução do código não entre em um loop infinito, também usamos nessa estrutura de código de uma variável de controle.

Repare que declaramos nossa estrutura condicional definindo que enquanto o valor de var for igual ou menor que 10, é exibido em tela por meio de nossa função print( ) o valor de var, em seguida atualizamos o valor de var somando ao seu valor atual uma unidade. Dessa forma o processo se repetirá até que var tenha o valor 10, encerrando o processo nesse ponto já que a condição definida foi alcançada.

O retorno será:

0
1
2
3
4
5
6
7
8

9
10

---

27 - Crie uma estrutura de repetição que percorre a string ‘Nikola Tesla’, exibindo em tela letra por letra desse nome:

```
for i in 'Nikola Tesla':
  print(i)
```

Sempre que estamos percorrendo o conteúdo de uma determinada variável que sabemos ter um número finito de elementos, podemos fazer a leitura de cada um destes elementos, um a um a cada laço de repetição, por meio de for.

Para a estrutura lógica do for é criada uma variável temporária (nesse caso “ i “) que a cada laço de repetição / a cada execução, irá ler um dos elementos atribuindo para si seu dado/valor, exibindo em tela seu conteúdo, até que tenha percorrido todos os elementos dessa variável.

Lembrando que o laço for consegue percorrer elementos de listas, dicionários e outras estruturas de dados parecidas, assim como em caso de uma string, caractere por caractere da mesma como se fosse um elemento individual.

Nesse caso o retorno será:

N
i
k
o
l
a

T
e
s
l
a

---

28 - Crie uma lista com 8 elementos de uma lista de compras de supermercado, por meio de um laço de repetição for liste individualmente cada um dos itens dessa lista.

```
lista10 = ['Água', 'Açúcar', 'Arroz', 'Pão', 'Leite', 'Massa', 'Carne', 'Refrigerante']

for i in lista10:
  print(i)
```

Quando estamos a percorrer os elementos de uma lista por meio de um laço for, cada elemento dentro de seu separador será lido individualmente, tendo seu conteúdo (independentemente do tipo de dado) associado a variável temporária i.

Nesse caso o retorno será:
Água
Açúcar
Arroz
Pão
Leite
Massa
Carne
Refrigerante

---

29 - Crie um programa que lê um valor de início e um valor de fim, exibindo em tela a contagem dos números dentro desse intervalo.

```
inicio = int(input('Digite o número onde começa a contagem: '))
fim = int(input('Digite o número onde termina a contagem: '))

for i in range(inicio, fim+1):
  print(i)
```

Sempre que temos um intervalo numérico com início e fim pré-estabelecidos, podemos usar do método range( ) para ler todos elementos desse intervalo. Associando o operador lógico in, usando range( ) dentro de um laço for, é possível percorrer e iterar sobre cada elemento dentro do intervalo.

Sendo assim, declaradas as variáveis inicio e fim que recebem números digitados pelo usuário, em nossa estrutura de repetição parametrizamos nosso método range( ) com os dados de inicio e de fim incrementado de 1. É necessário fazer essa pequena codificação adicional pois em Python quando estamos lendo números dentro de um intervalo, o último número lido não é contabilizado, mas serve como gatilho para que o processo seja encerrado naquele ponto.

Em outras palavras, apenas como exemplo, para percorrer elementos dentro de um intervalo entre 0 a 10 temos de parametrizar nosso range( ) com números entre 0 e 11.

Supondo que o usuário deu entrada dos números 20 e 30 o retorno será:

20
21
22
23
24
25
26
27

28
29
30

*Caso não houvéssemos definido fim+1 como parâmetro em range( ) a contagem encerraria em 29, não atendendo o enunciado da questão.

---

30 - Crie um programa que realiza a contagem de 0 a 20, exibindo apenas os números pares:

```
for i in range(0, 21):
  if i % 2 == 0:
    print(i)
```

Da mesma forma como no exemplo anterior, usando do método range( ) parametrizado com o valor de início e de fim (acrescido em uma unidade), podemos definir que serão exibidos apenas os números pares simplesmente criando uma condição onde apenas serão exibidos os números os quais o resto de sua divisão por 2 seja 0.

```
for i in range(0, 21, 2):
  print(i)
```

Uma forma alternativa que temos para resolver esse exercício é usando do terceiro parâmetro em justaposição do método range( ) que justamente define de quantos em quantos elementos devem ser retornados na função.

Como para nosso exemplo estamos exibindo números pares, que logicamente, sequencialmente são contados de dois em dois, parametrizando o método range( ) em seu terceiro parâmetro justaposto com o número 2, indiretamente exibiremos apenas os números pares.

O retorno será:

0
2
4
6
8
10
12
14
16
18

20

---

31 - Crie um programa que realiza a Progressão Aritmética de 20 elementos, com primeiro termo e razão definidos pelo usuário:

```
termo = int(input('Digite o primeiro termo: '))
razao = int(input('Digite a razão: '))
pa = termo + (20 -1) * razao

for i in range(termo, pa + razao, razao):
  print(i)
```

Lembrando que uma progressão aritmética é uma operação onde definimos um número inicial e uma constante, também chamados de termo e razão, respectivamente. A progressão em si nada mais é do que a soma do termo anterior com a constante.

Para nosso exercício pedimos que o usuário dê entrada tanto no termo (valor inicial) quanto na razão (constante) por meio da função input( ) atribuindo esses valores a suas respectivas variáveis.

Na sequência criamos uma variável de nome pa que contextualizando a fórmula de uma progressão aritmética, pega o valor de termo, somando com a constante multiplicada pela razão. Lembrando de realizar a subtração da constante em 1 para que tenhamos o gatilho para encerrar a progressão dentro do valor definido.

A partir daí, podemos simplesmente criar um laço for que usando do método range( ) define o valor de termo como valor inicial, pa + razão como valor final, exibindo no intervalo estipulado por razão. Sendo que a cada execução do laço é exibido em tela o valor da progressão por meio da função print( ).

Supondo que para termo o usuário tenha digitado 10, e para razão tenha digitado 3, o retorno será:

10
13
16
19
22
25

28
31
34
37
40
43
46
49
52
55
58
61
64
67

---

32 - Crie um programa que exibe em tela a tabuada de um determinado número fornecido pelo usuário:

```
x = int(input('Digite um Número: '))

for num in range(1, 11):
    print(f'{x} X {num} = {x * num}')
```

Para gerar uma simples tabuada podemos fazer o uso do método range( ), nos poupando serviço de realizar cada multiplicação do número fornecido por outro número.

Inicialmente criamos uma variável de nome x que recebe do usuário um número por meio da função input( ), validando o mesmo como do tipo int pois em uma tabuada básica não temos números com casas decimais.

Em seguida por meio de um laço for, usando do método range( ) percorreremos um intervalo de 1 a 10, a cada execução retornando um valor para num. Indentado para esse laço for simplesmente exibimos em tela que o determinado número, multiplicado pelo valor atual de num, resulta em um dado valor, nesse caso, o próprio valor de x multiplicado pelo valor de num.

Tendo o usuário digitado 7, o retorno será:

7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70

---

33 - Crie um programa que realiza a contagem regressiva de 20 segundos:

```
from time import sleep

for i in range(20, -1, -1):
  print(i)
  sleep(1)
print('Contagem Terminada!!!')
```

Para esse exercício, uma forma automatizada e otimizada de gerar um mecanismo de contagem regressiva é usando do módulo sleep da biblioteca time. Para isso, via comando from time import sleep importamos tal módulo e suas funcionalidades.

Na sequência, para gerar a contagem regressiva podemos usar de um laço for com o método range( ) de modo que o valor inicial da contagem é 20, o valor final definido em -1 fará a contagem decrescente até 0, sendo o último parâmetro, referente ao intervalo de números, definido também como negativo para que a contagem aconteça de forma decrescente.

Indentado ao bloco do laço for, a cada execução é exibida em tela o número referente ao passo, assim como chamando a função sleep( ) parametrizada em 1, definimos que a cada execução do laço haverá um tempo de espera de 1 segundo. Dessa forma, teremos não somente uma contagem de 20 para 0 mas tal contagem será exibida sincronizada de um em um segundo.

Fora do laço for, por meio da função print( ), exibimos uma mensagem quando a contagem regressiva é encerrada.

O retorno será:

20
19
18
17
16
15
14

13
12
11
10
9
8
7
6
5
4
3
2
1
0
Contagem Terminada!!!

---

34 - Crie um programa que realiza a contagem de 1 até 100, usando apenas de números ímpares, ao final do processo exiba em tela quantos números ímpares foram encontrados nesse intervalo, assim como a soma dos mesmos:

```
contador = 0
soma = 0

for i in range(1, 101):
  if i % 3 == 0:
    soma += i
    contador += 1

print(f'Foram encontrados {contador} números ímpares.')
print(f'A soma destes números é: {soma}!!!')
```

Para esse caso, novamente temos um problema envolvendo muito mais lógica do que estrutura de dados em si. Raciocine que ao mesmo tempo em que identificamos cada número ímpar precisamos o guardar em uma variável para que seja possível o usar na soma do total.

Para isso criamos duas variáveis, uma para controle (contador) e uma que fará a soma dos elementos (soma), ambas inicialmente zeradas pois serão incrementadas a cada laço de repetição.

Para o laço em si criamos um laço for que realiza a contagem de 1 até 100, retornando essa contagem para a variável temporária i. Dentro do bloco de código do laço for criamos uma estrutura condicional onde se o módulo da divisão de i por 3 deve ser igual a 0. Sempre que essa condição for verdadeira, a variável soma é atualizada somando para si o valor de i, da mesma forma, a variável contador recebe o incremento em uma unidade.

Por fim simplesmente exibimos em tela via função print( ) o que o enunciado da questão nos pede.

O retorno será:
Foram encontrados 33 números ímpares.
A soma destes números é: 1683!!!

---

35 - Crie um programa que pede ao usuário que o mesmo digite um número qualquer, em seguida retorne se esse número é primo ou não, caso não, retorne também quantas vezes esse número é divisível:

```
numero = int(input('Digite um número: '))
divisoes = 0

for i in range(1, numero + 1):
  if numero % i == 0:
    divisoes += 1

if divisoes == 2:
  print(f'{numero} é primo!!!')
  print(f'{numero} é divisível por 1 ou por {numero}')
else:
  print(f'{numero} não é primo!!!')
  print(f'{numero} é divisível {divisoes} vezes...')
```

Lembrando que um número primo é aquele que apenas é divisível por 1 ou por ele mesmo, temos de criar essa condição lógica para poder solucionar esse exercício.

Inicialmente por meio da função input( ) pedimos que o usuário digite um número, guardando esse valor na variável numero. Também é criada uma variável divisões que inicialmente está zerada.

Na sequência criamos um laço for que percorrerá de 1 até o número digitado pelo usuário + 1 via método range( ) retornando os valores para i a cada laço de repetição. Dentro do bloco referente ao nosso laço for criamos uma estrutura condicional onde se o módulo da divisão de número por i for igual a 0, divisoes recebe um incremento.

Por fim, podemos criar uma simples estrutura condicional onde se o último valor atribuído a divisoes for igual a 2, exibimos em tela a mensagem referente a um número primo, caso contrário, exibimos em tela a mensagem referente a um número não primo.

Supondo que o usuário digitou 97, o retorno será:

97 é primo!!!
97 é divisível por 1 ou por 97.

Supondo que o usuário tenha digitado 98, o retorno será:
98 não é primo!!!
98 é divisível 6 vezes...

---

36 - Crie um programa que pede que o usuário digite um nome ou uma frase, verifique se esse conteúdo digitado é um palíndromo ou não, exibindo em tela esse resultado.

```
frase = str(input('Digite uma palavra ou frase: ')).strip().upper()
palavras = frase.split()
caracteres = ''.join(palavras)
fraseinvertida = ''

for i in range(len(caracteres) -1, -1, -1):
  fraseinvertida += caracteres[i]

print(caracteres, fraseinvertida)

if fraseinvertida == caracteres:
  print('É um palíndromo!!!')
else:
  print('Não é um palíndromo...')
```

Um palíndromo, caso você não lembre, é quando temos um nome ou até mesmo uma frase que pode ser lida por igual da maneira regular ou de trás para frente.

Desse modo, ao pedir que o usuário dê entrada em uma palavra ou frase temos de fazer algumas validações, primeira delas, garantir que todo conteúdo digitado seja um texto, isso é feito aplicando o método .str( ).

Também aplicamos um .strip( ) para remover possíveis espaços sobrando tanto no começo quanto no fim do texto e um .upper( ) para converter todos seus caracteres para maiúsculo, caso contrário, o interpretador Python pode não reconhecer corretamente um palíndromo, já fazendo essas validações temos a garantia que todo e qualquer texto será lido da mesma forma.

Em seguida, para nossa variável palavra aplicamos sobre nossa frase o método .split( ) que por sua vez irá separar todos os caracteres que compõe nossa string. De modo parecido criamos uma variável de nome caracteres que como atributo tem o retorno do método join( ) por sua vez parametrizado com palavras de modo a juntar todos os espaços. Essa validação é necessária para que em

casos de frases, os espaços entre as palavras também sejam removidos.

Dando sequência, é criada uma variável de nome fraseinvertida que inicialmente só tem como atributo uma string vazia.

Na sequência por meio de um laço for, usando de range( ) percorremos cada elemento de nossa variável caracteres de trás para frente, atualizando nossa variável fraseinvertida com cada caractere lido, preenchendo a mesma de forma que teremos os mesmos caracteres de caracteres mas em forma espelhada.

Por fim, por meio de uma estrutura condicional simples, definimos que se o conteúdo de fraseinvertida for igual ao de caracteres, exibiremos em tela que neste caso temos um palíndromo, caso contrário, retornamos que não.

Supondo que o usuário tenha digitado ‘Socorram me subi no onibus em Marrocos’, o retorno será:
SOCORRAMMESUBINOONIBUSEMMARROCOS
SOCORRAMMESUBINOONIBUSEMMARROCOS
É um palíndromo!!!

Supondo que o usuário tenha digitado ‘Fernando Feltrin’, o retorno será:
FERNANDOFELTRIN NIRTLEFODNANREF
Não é um palíndromo...

---

37 - Declare uma variável que por sua vez recebe um nome digitado pelo usuário, em seguida, exiba em tela uma mensagem de boas vindas que incorpora o nome anteriormente digitado, fazendo uso de f'strings.

```
nome = input('Digite o seu nome: ')

print(f'Bem-Vindo(a) {nome}, é um prazer te receber em nosso Hotel.')
```

Quando queremos usar de strings mais elaboradas, não somente texto mas podendo agregar/integrar a esse texto diversas funcionalidades, usamos do conceito de interpolação, na sintaxe mais moderna da linguagem Python, chamada de f'strings e máscaras de substituição.

Ao declarar f' o interpretador Python já identifica todo e qualquer conteúdo subsequente como uma string "com super poderes", podendo a qualquer momento usar de uma máscara de substituição representada por " { } " chaves, sendo que dentro das mesmas pode ser inserido todo e qualquer conteúdo, desde o conteúdo de uma variável até métodos de classe instanciando objetos e realizando funções.

Em nosso exemplo, na f'string posta como parâmetro de nossa função print( ) no lugar da máscara {nome} será colocado o conteúdo da variável nome.

Supondo que o usuário digitou Maria, o retorno será:
Bem-Vindo(a) Maria, é um prazer te receber em nosso Hotel.

---

38 - Peça para que o usuário digite um número, diretamente dentro da função print( ) eleve esse número ao quadrado, exibindo o resultado incorporado a uma mensagem:

```
num = int(input('Digite um número: '))

print(f'{num} elevado ao quadrado resulta {num ** 2}')
```

Novamente, usando de f'strings e máscaras de substituição podemos criar interações entre uma string, dados e operações, até mesmo usando desses dados diretamente em funções de saída como nossa velha conhecida print( ).

Nesse caso, é recebido do usuário um número por meio da função input( ), atribuindo esse dado/valor a variável num.

Na sequência, em nossa função print( ) usando de f'strings, na primeira máscara de substituição será instanciada o conteúdo da variável num, assim como na segunda máscara estaremos realizando a operação de elevar o valor de num ao quadrado.

Supondo que o usuário tinha digitado 8, o resultado será:
8 elevado ao quadrado resulta 16

---

39 - Dada a seguinte lista: nomes = [‘Ana’, ‘Carlos’, ‘Daiane’, ‘Fernando’, ‘Maria’], substitua o terceiro elemento da lista por ‘Jamile’:

```
nomes = ['Ana', 'Carlos', 'Daiane', 'Fernando', 'Maria']

nomes[2] = 'Jamile'

print(nomes)
```

Para substituir de forma permanente um determinado elemento de uma lista, precisamos atualizar a variável a qual a lista está associada. Sendo assim, fazendo referência a variável nomes em sua posição de índice 2 estaremos trabalhando com o terceiro elemento da lista, atribuindo para este elemento a nova string ‘Jamile’, ao exibir em tela o conteúdo dessa variável teremos de fato a lista com esta alteração.

O retorno será:
[‘Ana’, ‘Carlos’, ‘Jamile’, ‘Fernando’, ‘Maria’]

---

40 - Adicione o elemento ‘Paulo’ na lista nomes:

```
nomes = ['Ana', 'Carlos', 'Jamile', 'Fernando', 'Maria']

nomes.append('Paulo')

print(nomes)
```

Para simplesmente adicionarmos um novo elemento a uma lista já existente, usamos do método append( ) parametrizado com o dado/valor a ser adicionado. Lembrando que usando do método append( ) o elemento em questão sempre será adicionado como último elemento da lista.

O retorno será:
[‘Ana’, ‘Carlos’, ‘Jamile’, ‘Fernando’, ‘Maria’, ‘Paulo’]

---

41 - Adicione o elemento ‘Eliana’ na lista nomes, especificamente na terceira posição da lista:

```
nomes = ['Ana', 'Carlos', 'Jamile', 'Fernando', 'Maria', 'Paulo']

nomes.insert(2, 'Eliana')

print(nomes)
```

Quando necessitamos incluir um novo elemento em uma posição específica da lista, temos de fazer o uso do método insert( ), parametrizando o mesmo com o número do índice a ser usado, seguido do dado/valor a ser inserido naquela posição.

O retorno será:
[‘Ana’, ‘Carlos’, ‘Eliana’, ‘Jamile’, ‘Fernando’, ‘Maria’, ‘Paulo’]

---

42 - Remova o elemento ‘Carlos’ da lista nomes:

```
nomes = ['Ana', 'Carlos', 'Jamile', 'Fernando', 'Maria', 'Paulo']

nomes.remove('Carlos')

print(nomes)
```

Para remover um elemento específico de uma lista, podemos usar do método remove( ) por sua vez parametrizado com o nome do elemento. Caso o elemento não exista entre os elementos que compõe a lista, será gerado um erro “x not in list” pois o elemento não consta na lista.

O retorno será:
[‘Ana’, ‘Eliana’, ‘Jamile’, ‘Fernando’, ‘Maria’, ‘Paulo’]

---

43 - Mostre o segundo, terceiro e quarto elemento da lista nomes. Separadamente, mostre apenas o último elemento da lista nomes:

```
nomes = ['Ana', 'Carlos', 'Jamile', 'Fernando', 'Maria', 'Paulo']

print(nomes[1:4])

print(nomes[-1])
```

Para instanciar e exibir determinados elementos de uma lista, basta que façamos referência a seus respectivos números de índice.

Nesse caso, para exibir o segundo, terceiro e quarto elementos (em outras palavras do segundo ao quarto elemento) podemos instanciar a variável nomes[ ] fazendo referência aos índices 1:4 (lembrando que em Python todo e qualquer índice começa em 0, assim como o último elemento lido não é exibido, servindo internamente apenas como gatilho para encerramento de contagem de elementos por parte do interpretador).

De forma parecida, para instanciar e exibir especificamente apenas o último elemento de uma lista, usamos a notação -1 como referência de índice.

O retorno será:
[‘Carlos’, ‘Jamile’, ‘Fernando’]
Paulo

---

44 – Crie um dicionário via método construtor dict( ), atribuindo para o mesmo ao menos 5 conjuntos de chaves e valores representando objetos e seus respectivos preços:

```
dicionario = dict(Pão = 'R$ 1,99',
                  Açúcar = 'R$ 4,99',
                  Café = 'R$ 3,49',
                  Macarrão = 'R$ 3,99',
                  Carne = 'R$ 16,99')

print(dicionario)
```

Para criar um dicionário em Python usando de seu método construtor, inserindo dados manualmente um a um, é necessário que se respeite algumas características.

Inicialmente todo e qualquer dicionário deve estar atribuído a uma variável, logo após usando do método dict( ) nosso interpretador espera que os dados que serão repassados como parâmetro irão compor os campos de um dicionário, em outras palavras, suas chaves e valores.

Por fim, ao inserir dados de forma manual, no campo de parâmetros devemos colocar tais dados como se estivéssemos declarando variáveis, ou seja, dando um nome, usando do operador de atribuição " = " seguido do atributo, porém, internamente o que o método construtor de dicionários fará é converter nossa "variável" em chave, assim como seu "atributo" em valor.

Em nosso exemplo, apenas pegando um dos itens para exemplificar, Café = 'R$ 3,49' será convertido em 'Café':'R$ 3,49', sendo Café a chave e R$ 3,49 o valor deste item no dicionário.

Por meio da função print( ), parametrizada com o conteúdo da variável dicionário, podemos ver o dicionário em si em sua estrutura.

O retorno será:
{'Pão' : 'R$ 1,99', 'Açúcar' : 'R$ 4,99', 'Café' : 'R$ 3,49', 'Macarrão' : 'R$ 3,99', 'Carne' : 'R$ 16,99'}

---

45 – Crie um dicionário usando o construtor de dicionários do Python, alimente os valores do mesmo com os dados de duas listas:

```
itens = ['Caneta', 'Lápis', 'Borracha', 'Caderno']
precos = ['1,99', '0,99', '0,50', '9,90']

dicionario1 = dict(keys = itens, values = precos)

print(dicionario1)
print(type(dicionario1))
```

Quando estamos trabalhando com o tipo de dado dicionário em Python temos algumas particularidades, primeira delas que um dicionário pode ser criado diretamente pela sintaxe de todo e qualquer conteúdo disposto entre { } chaves, assim como usar do método construtor dict( ) onde o que é passado como parâmetro para este método se torna dados de um dicionário.

Importante levar em consideração que um dicionário obrigatoriamente possui seus dados dispostos em "chave : valor" sendo assim, quando criamos um dicionário via método construtor devemos inserir estes tipos de dados em seus respectivos campos.

De acordo com o enunciado, criamos as listas, em seguida criamos uma variável de nome dicionario1 que por sua vez usando do método construtor dict( ) repassa para keys os dados da lista itens, da mesma forma repassando para values os dados da lista precos.

Via função print( ), parametrizada com o conteúdo de dicionário1 e na sequência parametrizada com o tipo da variável dicionario1, podemos gerar um retorno.

O retorno será:
{'keys':['Caneta', 'Lápis', 'Borracha', 'Caderno'], 'values':['1,99', '0,99', '0,50', '9,90']}
<class 'dict'>

---

46 - Crie uma simples estrutura de dados simulando um cadastro para uma loja. Nesse cadastro deve conter informações como nome, idade, sexo, estado civil, nacionalidade, faixa de renda, etc... Exiba em tela tais dados:

```
cadastro = {'Nome':'Paulo',
            'Sexo':'Masculino',
            'Idade':38,
            'Nacionalidade':'Brasileiro',
            'Estado Civil':'Divorciado',
            'Escolaridade':['Ensino Superior',
                            'Doutorado'],
            'Ocupação':'Professor',
            'Renda':'1999.00 - 2999.00'}

print(cadastro)
```

Nesse exercício temos a orientação de criar uma estrutura de dados onde certos campos devem guardar certas informações, o que condiz perfeitamente com a estrutura de dado de um dicionário em Python, onde podemos atribuir a uma variável uma série de dados organizados em chaves e valores.

Lembrando que pela sintaxe um dicionário é uma estrutura identificada por " { } " chaves, sendo seus dados internos organizados em " chave : valor " aceitando todo e qualquer tipo de dado para esses campos, inclusive dicionários dentro de dicionários, respeitando sempre a sintaxe dos tipos de dados usados.

Para cada campo criado de acordo com o enunciado da questão, respeitando a sintaxe, criamos um preenchimento com dados fictícios, atendendo todos os requisitos da questão.

O retorno será:
{'Nome':'Paulo', 'Sexo':'Masculino', 'Idade':38, 'Nacionalidade':'Brasileiro', 'Estado Civil':'Divorciado', 'Escolaridade': ['Ensino Superior','Doutorado'], 'Ocupação':'Professor', 'Renda':'1999.00 – 2999.00'}

---

47 - Crie um programa que recebe dados de um aluno como nome e suas notas em supostos 3 trimestres de aula, retornando um novo dicionário com o nome do aluno e a média de suas notas:

```
aluno = [{'Nome':'Fernando', 'Notas': [62, 73, 90]}]

def calcula_media(aluno):
  notas = []
  for media in aluno:
    if len(media['Notas']) > 0:
      temp = round(sum(media['Notas'])/len(media['Notas']))
    else:
      temp = 0
    notas.append({'Nome':media['Nome'],'Média das notas':temp})
  print(notas)

media_estudante = calcula_media(aluno)
```

Para um exercício como esse, teremos de por meio de uma função desempacotar os dados de um aluno, realizando algumas operações e validações para o mesmo, por fim retornando um novo dicionário.

Para isso, inicialmente analisando nossa variável aluno, podemos notar que temos dicionário e lista dentro de lista, de modo que temos que temos que iterar sobre chaves e valores do dicionário, de modo a separar o campo referente as notas para que possamos realizar o cálculo da média das mesmas.

Sendo assim, criamos uma função de nome calcula_media( ) que recebe os dados de aluno como parâmetro. No bloco da função criamos uma variável de nome notas que inicialmente possui como atributo uma lista vazia. Em seguida, por meio de um laço for percorremos os dados de aluno, mais especificamente onde consta o campo 'Notas', validando que caso seu tamanho seja maior que 0, é realizado a divisão entre a soma dos valores de notas pelo tamanho de notas. Em outras palavras, para realizar a média, pegamos todos os valores, somamos os mesmos, dividindo pelo

número de trimestres nesse caso, retornando esses dados para temp.

Caso essa condição (número de notas disponíveis no trimestre) não for verdadeira, a variável temp recebe o valor zero.

Finalizando esse bloco, a variável notas é atualizada por meio do método append( ), recebendo em forma de dicionário o nome do aluno e a média das notas guardadas anteriormente na variável temp.

Por fim, é exibido em tela o conteúdo de notas.

O retorno será:
{‘Nome’: ‘Fernando’, ‘Média das notas’: 75}

---

48 - Crie um sistema de perguntas e respostas que interage com o usuário, pedindo que o mesmo insira uma resposta. Caso a primeira questão esteja correta, exiba em tela uma mensagem de acerto e parta para a próxima pergunta, caso incorreta, exiba uma mensagem de erro e pule para próxima pergunta:

```
base = {
    'Pergunta 01 ':{
        'pergunta':' Quanto é 4 X 4 ? ',
        'alternativas':{'a':'12','b':'24','c':'16','d':'20'},
        'resposta_certa':'c',
    },
    'Pergunta 02 ':{
        'pergunta':' Quanto é 6 / 3 ? ',
        'alternativas':{'a':'2','b':'1','c':'3','d':'4'},
        'resposta_certa':'a',
    },
}

respostas_certas = 0

for pkeys, pvalues in base.items():
  print(f'{pkeys}:{pvalues["pergunta"]}')

  for rkeys, rvalues in pvalues['alternativas'].items():
    print(f'[{rkeys}]: {rvalues}')

  resposta = input('Escolha uma alternativa: [a],[b],[c] ou [d]')

  if resposta == pvalues['resposta_certa']:
    print('Resposta Correta!!!')
    respostas_certas += 1
  else:
    print('Resposta Incorreta!!!')

if respostas_certas == 0:
  print('Você não acertou nenhuma questão.')
elif respostas_certas == 1:
  print('Você acertou apenas uma questão.')
else:
  print('Você acertou todas as questões.')
```

De acordo com o enunciado da questão, uma vez que temos uma base de dados de perguntas e respostas, podemos associar que essa base pode ser criada com um dicionário onde suas chaves e valores se tornam perguntas e respostas, respectivamente.

Raciocine que como teremos mais de uma pergunta de múltipla escolha, podemos abstrair essa ideia para um dicionário dentro de outro dicionário, importante raciocinar também que para cada pergunta teremos a pergunta em si, as alternativas e a resposta, para que possamos assim interagir com o usuário, verificando se o mesmo acertou ou errou em suas respostas.

Inicialmente criamos uma variável de nome base, que por sua vez tem um dicionário onde a primeira chave é 'Pergunta 01 ', sendo seu valor outro dicionário com a pergunta, as alternativas dentro de outro dicionário e a resposta certa para essa questão. O mesmo é feito para 'Pergunta 02' e poderia ser replicado para quantas perguntas fossem necessárias.

Na sequência declaramos a variável respostas_certas, inicialmente com valor zero. Em seguida criamos um laço for que desempacotará dos itens de nossa base as chaves e valores referentes as perguntas. Dentro desse mesmo bloco é criado outro laço for para que percorra da pergunta, quais chaves e valores correspondem as alternativas.

Uma vez criada a estrutura que exibirá em tela as perguntas e suas respectivas alternativas, podemos pedir que o usuário dê entrada na alternativa o qual considera a correta de acordo com a pergunta.

Na sequência criamos uma estrutura condicional onde se o valor de resposta for igual ao valor situado no campo 'resposta_certa' de nossa pergunta, é exibido em tela uma mensagem de acerto, também atualizando o valor da variável respostas_certas em 1 unidade. Caso contrário, é simplesmente exibida em tela uma mensagem de erro.

Por fim, fora do laço for, novamente no escopo global de nosso código, criamos uma simples estrutura condicional que com base no valor de respostas_certas exibe em tela o número de questões que o usuário acertou.

Nesse caso, supondo que para a primeira questão o usuário tenha digitado ‘ d ‘ e para a segunda questão tenha digitado ‘ a ‘, o retorno será:
Resposta Incorreta!!!
Resposta Correta!!!
Você acertou apenas uma questão.

---

49 - Crie uma função de nome funcao1, que por sua vez não realiza nenhuma ação:

```
def funcao1():
  pass
```

Em Python podemos declarar uma função qualquer, mesmo sem um propósito definido inicialmente, para isso definimos a função como de costume, respeitando sua sintaxe, e no corpo da função simplesmente colocamos a palavra reservada pass, que internamente dá a instrução ao interpretador para que essa linha, ou bloco de código, seja ignorado.

Nesse caso, nenhum retorno é gerado.

---

50 - Atribua a função funcao1 a uma variável:

```
def funcao1():
  pass

var1 = funcao1()
```

Após criada a estrutura de uma função qualquer, podemos declarar uma variável que tem atribuída para si uma função. Dessa forma, quando fizermos o uso dessa variável estaremos instanciando e inicializando sua função. No jargão popular, a variável var1 está chamando a função funcao1( ).

Nesse caso, nenhum retorno é gerado.

---

51 - Crie uma função que retorna um valor padrão:

```
def msg():
  return 0

msg()
```

Uma função pode ou não retornar algum dado/valor para atualizar uma variável ou apenas para controle. Em muitas linguagens de programação existe a convenção de que quando um script/código é executado sem erros, o mesmo pode internamente retornar o valor 0, assim como quando houverem erros pode retornar o valor 1.

Sendo assim, em Python podemos perfeitamente dentro de uma função criar um retorno para controle, apenas para verificar se o bloco de código da função foi executado corretamente assim como em estruturas mais robustas podemos usar do próprio "resultado" do processamento da função para atualizar algum elemento do escopo global do código como uma variável por exemplo.

O retorno será:
0

---

52 - Crie uma função que exibe em tela uma mensagem de boas-vindas:

```
def mensagem():
  print('Bem Vindo(a)!!!')

var1 = mensagem()
```

Uma vez definida a função mensagem( ), sem parâmetros mesmo, em seu corpo colocamos uma função print( ) por sua vez parametrizada com a string 'Bem Vindo(a)!!!'.

Na sequência declaramos uma variável de nome var1 que instancia e inicializa a função mensagem( ), dessa forma, ao executar nosso código, nossa variável var1 irá chamar a função mensagem( ) que por sua vez realizará sua função programada, nesse caso, exibir em tela a mensagem parametrizada em print( ).

O retorno será:
Bem Vindo(a)!!!

---

53 - Crie uma função que recebe um nome como parâmetro e exibe em tela uma mensagem de boas-vindas. O nome deve ser fornecido pelo usuário, incorporado na mensagem de boas-vindas da função:

```
def mensagem(nome):
  print(f'Bem vindo(a) {nome}')

nome = input('Digite o seu nome: ')
nome = mensagem(nome)
```

Inicialmente definimos nossa função mensagem( ) que receberá algum dado/valor a ser usado como parâmetro em nome. Dentro do corpo da função usamos da função print( ) e de f'strings para criar uma mensagem de boas-vindas onde em uma máscara de substituição { } será instanciado o nome do parâmetro nome.

Em seguida simplesmente criamos uma variável nome que por meio da função input( ) pede que o usuário digite o mesmo.

Por fim, a variável nome chama a função mensagem( ) passando como parâmetro o conteúdo da variável nome.

Supondo que o usuário tenha digitado 'Fernando', o retorno será:
Bem vindo(a) Fernando

---

54 - Crie uma função que recebe um valor digitado pelo usuário e eleva esse valor ao quadrado:

```
def exp(num):
  return num ** 2

num = int(input('Digite um número: '))
num = exp(num)

print(num)
```

*Inicialmente definimos uma função de nome exp( ) que recebe como parâmetro um número em num. Dentro do corpo da função retornamos o valor de num elevado ao quadrado, por meio da expressão num ** 2.*

Na sequência pedimos que o usuário digite um número, validando o mesmo como do tipo int, apenas por convenção, atribuindo esse valor a variável num.

Em seguida a variável num chama a função exp( ) parametrizando a mesma com o próprio valor. Por fim, via função print( ) exibimos em tela o valor de num.

Lembrando que como aqui estamos usando a variável num inicialmente para guardar o valor digitado pelo usuário, e posteriormente chamando a função, estamos atualizando o próprio valor de num com o retorno da função.

Supondo que o usuário tenha digitado 8, o retorno será:
64

---

55 - Crie uma função com dois parâmetros relacionados ao nome e sobrenome de uma pessoa, a função deve retornar uma mensagem de boas-vindas e esses dados devem ser digitados pelo usuário:

```
def boas_vindas(nome, sobrenome):
  print(f'Bem vindo(a) {nome} {sobrenome}')

nome = input('Digite o seu nome: ')
sobrenome = input('Agora digite o seu sobrenome: ')

pessoa = boas_vindas(nome, sobrenome)
```

Se tratando de funções em Python, uma função pode não ter nenhum parâmetro, assim como ter um ou mais parâmetros desde que eles tenham um propósito explícito no corpo da função.

Nessa linha de raciocínio, definimos nossa função boas_vindas( ) que por sua vez receberá dois parâmetros, nome e sobrenome. No corpo da função por meio da função print( ) usaremos de uma f'string para incorporar na mensagem de boas-vindas os dados oriundos dos parâmetros nome e sobrenome, respectivamente.

Na sequência criamos as estruturas de interação com o usuário, por meio da função input( ), pedindo que o mesmo digite seu nome e sobrenome, atribuindo esses dados as variáveis nome e sobrenome.

Por fim declaramos uma variável de nome pessoa, que chama a função boas_vindas( ) parametrizando a mesma em nome com o conteúdo da variável nome, da mesma forma parametrizando a mesma em sobrenome com o conteúdo da variável sobrenome.

Supondo que o usuário tenha digitado 'Fernando' e 'Feltrin', o retorno será:
Bem vindo(a) Fernando Feltrin

---

56 - Crie uma função com dois parâmetros, sendo um deles com um dado/valor predeterminado:

```
def boas_vindas(nome, nacionalidade = 'Brasileiro'):
  print(f'{nome} é {nacionalidade}!!!')

nome = input('Digite o seu nome: ')
ex1 = boas_vindas(nome)

nacionalidade = input('Digite sua nacionalidade: ')
ex2 = boas_vindas(nome, nacionalidade)
```

Se tratando do modo como parametrizamos uma função, em Python temos a flexibilidade de realizar algumas ações diretamente sobre os parâmetros. Uma delas é no momento da declaração dos parâmetros/argumentos de uma função, definir um valor padrão, pré-determinado. Isso é muito útil para casos onde precisamos de um determinado parâmetro fornecido pelo usuário, porém o mesmo não dá entrada neste dado, assim contornamos um possível erro a ser gerado.

Em outras palavras, ao definirmos um parâmetro com um valor padrão, caso o usuário dê entrada em um dado/valor para esse parâmetro, o mesmo será atualizado com esse dado/valor fornecido pelo usuário, caso contrário, usará do valor padrão pré-determinado.

Aqui, usando do mesmo exemplo anterior, em nossa função boas_vindas( ) temos dois parâmetros, o primeiro deles, nome, da forma como está declarado é obrigatório que tenha um dado aqui repassado por uma variável ou pelo usuário, já o segundo parâmetro, que agora tem o nome nacionalidade, possui como padrão 'Brasileiro'.

No corpo da função é criada uma f'string que incorpora tanto os dados de nome quanto de nacionalidade em uma mensagem a ser exibida em tela por meio da função print( ).

Na sequência, apenas para exemplo, criamos duas estruturas que receberão dados do usuário via input( ) guardando esses dados para as variáveis nome e nacionalidade.

Em nossa variável ex1 chamamos a função boas_vindas( ) parametrizando a mesma apenas com nome. Já para nossa variável ex2, também chamamos a função boas_vindas( ), dessa vez parametrizando a mesma com os dados de nome e de nacionalidade.

Supondo que o usuário tenha digitado ‘Maria’ e ‘Portuguesa’, o retorno para ex1 será:
Maria é Brasileiro.
Enquanto o retorno para ex2 será:
Maria é Portuguesa.

Como mencionado anteriormente, em ex1 não repassamos o parâmetro referente à nacionalidade, então foi usado o padrão ‘Brasileiro’, em ex2 repassamos tanto os dados de nome quanto de nacionalidade, dessa forma o parâmetro nacionalidade da função boas_vindas( ) teve seu dado/valor atualizado de ‘Brasileiro’ para ‘Portuguesa’.

---

57 - Crie uma função com três parâmetros, sendo dois deles com dados/valores padrão, alterando o terceiro deles contornando o paradigma da justaposição de argumentos:

```
def pessoa1(nome, idade = 18, funcao = 'técnico'):
  print(f'{nome} tem {idade} anos, e sua função é {funcao}')

p1 = pessoa1('Fernando', funcao = 'gerente')
```

Por justaposição entendemos que certos argumentos, quando passados de forma simples, obedecerão a ordem ao qual os mesmos forem inseridos. Em outras palavras, ao chamar uma função, o primeiro parâmetro repassado será instanciado para o primeiro parâmetro da função, o segundo repassado para o segundo parâmetro declarado da função, e assim sucessivamente.

Quando queremos contornar a justaposição, repassando um determinado dado/valor para um parâmetro de função em uma posição específica, podemos simplesmente instanciar o parâmetro o qual queremos repassar o dado na própria chamada da função.

Em nosso exemplo temos a função pessoa1( ) que receberá um nome, uma idade (tendo um argumento pré-definido 18) e uma função (tendo seu argumento pré-definido como 'técnico'). A função em si quando executada apenas exibe em tela uma f'string incorporando os parâmetros em uma mensagem.

Por fim declaramos uma variável de nome p1 que instancia e inicializa a função pessoa1( ) parametrizando a mesma com 'Fernando' que por justaposição será repassado para nome, e função = 'gerente' que nesse caso atualiza o parâmetro função de 'técnico' para 'gerente'. Como ao chamar a função não foi repassado nenhum dado para idade, a função usará do valor pré-determinado 18.

Nesse caso o retorno será:
Fernando tem 18 anos, e sua função é gerente

---

58 - Crie uma função que pode conter dois ou mais parâmetros, porém sem um número definido e declarado de parâmetros:

```
def msg(*args):
  print(f'Os parâmetros são: {args}')

ex2 = msg('nome = Fernando', 'idade = 33', 'profissão = professor')
```

*Em algumas situações bastante específicas pode ser que tenhamos de definir uma função onde inicialmente não sabemos ao certo quais e quantos parâmetros teremos para a mesma. Uma maneira de contornar esse problema, ao menos para definir a função e realizar alguns testes com a mesma, é usar do marcador *args. Com este tipo de marcador, podemos criar quantos parâmetros quisermos temporariamente, os parâmetros por sua vez serão justapostos e estarão usáveis pelo corpo da função no momento de sua execução.*

*Para ficar mais claro, vamos ao exemplo, onde inicialmente definimos a função msg( ) por sua vez parametrizada com *args. No corpo dessa função temos uma simples mensagem a ser exibida em tela pela função print( ) onde exibiremos tudo o que for repassado para args. Note que o marcador " * " é usado apenas no campo do parâmetro na declaração da função, apenas para que o interpretador reconheça que ali serão inseridos parâmetros posteriormente.*

Na sequência criamos uma variável de nome ex2 que chama a função msg( ) parametrizando a mesma com uma série de parâmetros, separados por vírgula, nesse caso apenas para fins de exemplo, simulando parâmetros já com dados/valores pré-determinados. Dessa forma, como parâmetros temporários para essa função teremos 'nome = Fernando', 'idade = 33' e 'profissão = professor'.

Lembrando mais uma vez que esse tipo de parâmetro é temporário, após executar uma vez essa função, esses dados serão descarregados da memória e a função terá de ser parametrizada novamente.

O retorno será:
Os parâmetros são: (‘nome = Fernando’, ‘idade = 33’, ‘profissão = professor’)

---

59 - Crie uma função de número de parâmetros indefinido, que realiza a soma dos números repassados como parâmetro, independentemente da quantidade de números:

```
def soma(*args):
  num = 0
  for valordigitado in args:
    num += valordigitado
  print(f'O resultado da soma é: {num})

soma = soma(18, 43, 99, 1)
```

*No exemplo anterior foi mencionado que quando fazemos o uso de *args, os dados/valores repassados para esse campo se tornam parâmetros "temporários" pois uma vez que os mesmos tenham sido usados pela função são descarregados da memória e a função volta ao seu estado inicial sem parâmetros até que novamente seja parametrizada.*

*Para realizarmos uma interação mais interessante com os parâmetros que estamos usando em *args, além de simplesmente os exibir em tela, é necessário que no corpo da função realizemos o "desempacotamento" dos mesmos, atribuindo seus dados/valores a uma ou mais variáveis. Dessa forma temos dados repassados como parâmetros e de fato usáveis dentro do bloco de código da função.*

*De acordo com o que é pedido no enunciado da questão, para criar uma calculadora de soma, que realiza a soma de quantos números forem necessários, inicialmente definimos nossa função com o marcador *args.*

*Dentro do corpo dessa função, criamos uma variável de nome num, inicialmente de valor 0, a ser atualizada com dados oriundos de *args.*

*Para desempacotar os dados de *args, podemos utilizar de um laço for, que a cada laço de repetição lê o valor em *args salvando seu valor em uma variável, nesse caso, para uma simples calculadora de soma, atualizando o valor de num e somando seu*

*valor atual com cada valor novo lido em *args pelo laço for atribuído a variável temporária valordigitado.*

Por fim, para testar se nossa função soma( ) cumpre o que é esperado para a mesma, criamos uma variável de nome soma que por sua vez chama a função soma, passando como parâmetros os números 18, 43, 99 e 1.

O retorno será:
O resultado da soma é: 161

---

60 - Crie uma função que recebe parâmetros tanto por justaposição (*args) quanto nomeados (kwargs):**

```
def identificacao(*args, **kwargs):
  for n in args:
    nome = n
    #print(f'{n}')

    for k, v in kwargs.items():
      idade = k
      sexo = v
      #print(f'{idade}, {sexo}')

      print(f'Nome: {nome},  {idade},  {sexo}')

pessoa = identificacao('Fernando', idade = 33, sexo = 'M')
```

*Este é um exercício interessante pois coloca à prova diferentes níveis de desempacotamento de dados de *args e **kwargs, de modo que para conseguirmos realizar as devidas interações com os parâmetros, é de suma importância gerar a cadeia de interações na ordem e indentação correta.*

*De acordo com o enunciado da questão, teremos de fazer uso tanto de número de parâmetros indefinido, porém justapostos, quanto de parâmetros nomeados. Para isso, definimos nossa função identificacao( ) que recebe como parâmetros *args e **kwargs.*

*Apenas fazendo um adendo, a nomenclatura 'args' e 'kwargs' pode ser substituída por qualquer outro nome, como *números e **nomes por exemplo, o marcador em si que o interpretador Python identifica na verdade são os " * " e " ** ". Outro ponto que não havia sido comentado é que quando usamos de *args e **kwargs, por convenção usamos desses marcadores nessa ordem, nunca **kwargs e *args pois em algumas IDEs essa ordem pode gerar exceções.*

*Dando sequência com nosso código, para guardar o dado/valor de *args simplesmente usamos de um laço for que lê esse argumento, o salvando em uma variável.*

*Dentro desse laço criaremos um novo laço de repetição, agora com as variáveis temporárias k e v que percorrerão os argumentos em *kwargs.items( ) uma vez que os dados estarão em formato de dicionário, guardando esses dados/valores em suas respectivas variáveis.*

Por fim, podemos usar de nossa velha conhecida função print( ) para exibir em tela tais dados.

*Fora do corpo de nossa função, no escopo global do código, declaramos uma variável de nome pessoa que chama a função identificacao( ) parametrizando a mesma com 'Fernando', que por justaposição será atrelado ao campo *args, seguido de dois outros parâmetros nomeados idade e sexo, que justamente por serem nomeados irão gerar e compor os campos em *kwargs.*

O retorno será:
Nome: Fernando, idade, 33
Nome: Fernando, sexo, M

---

61 - Escreva um programa que retorna o número de Fibonacci: Sendo o número de Fibonacci um valor iniciado em 0 ou em 1 onde cada termo subsequente corresponde à soma dos dois anteriores.

```
def fibonacci(n):
 if n <= 1:
   return n
 else:
   return fibonacci(n-1) + fibonacci(n-2)

num = int(input('Digite um numero para encontrar seu Fibonacci: '))
resposta = fibonacci(num-1)
print(resposta)
```

Inicialmente criamos nossa função fibonacci( ) que receberá como parâmetro um número. Indentado dentro do bloco da função criamos a estrutura lógica para gerar um número Fibonacci. Por meio de uma estrutura condicional iniciamos validando que caso o valor de n for igual ou menor que 1 é retornado o próprio número pois não temos termos suficientes para realizar as devidas somas. Caso essa condição não seja válida, então a função retorna uma operação de soma instanciando a própria função fibonacci( ), uma vez parametrizada com n – 1 e também parametrizada com n – 2.

Em seguida criamos uma variável de nome num que recebe via input( ) um número digitado pelo usuário, validado como inteiro.

Também criamos uma variável de nome resposta, que chama a função parametrizando a mesma com o número fornecido pelo usuário salvo em num – 1, finalizando o processo exibindo em tela o resultado da operação.

Supondo que o usuário tenha digitado 8, o retorno será:
13

---

62 - Crie um programa modularizado, onde em um arquivo teremos uma lista de médicos fictícios a serem consultados, em outro arquivo, teremos a estrutura principal do programa, que por sua vez realiza o agendamento de uma consulta médica com base na interação com o usuário.

```
# Arquivo medicos.py
medicos = ['Grazielle Veiga',
           'Matheus Correa']

# Arquivo main.py
import medicos

menu = str(input('Deseja agendar uma consulta? (S ou N) ')).upper()

if menu == 'S':
  paciente = input('Por favor, digite seu nome completo: ')
  print(f'{paciente}, escolha com qual médico deseja consultar:')
  print('1 - Grazielle Veiga')
  print('2 - Matheus Correa')
  medico = int(input('Com qual médico deseja agendar consulta?'))
  if medico == 1:
    print(f'Sua consulta com a Dr.a {medicos.medicos[0]} será agendada.')
  if medico == 2:
    print(f'Sua consulta com o Dr. {medicos.medicos[1]} será agendada.')
else:
  print('Agradecemos o seu contato!!!')
```

Quando estamos trabalhando com modularização, subentendemos que os módulos os quais estaremos fazendo uso contém blocos de código isolados que podem ser utilizados por nosso programa. Uma prática muito comum em programas com muitas funcionalidades é que grande parte das mesmas sejam modularizadas, isto nos proporciona uma performance melhor de nosso programa pois no lugar de carregar tudo o que está disponível para o mesmo, importamos apenas o necessário conforme a demanda do programa. Modularização também é muito importante para a manutenção de um programa, pois podemos implementar ou alterar funcionalidades sem que um erro na parte modularizada afete o funcionamento do programa como um todo.

Pois bem, de acordo com o enunciado então inicialmente criamos um arquivo .py de nome médicos, dentro do mesmo a variável médicos possui uma lista com nomes de médicos fictícios.

Em nosso arquivo principal main.py, inicialmente realizamos a importação do módulo médicos por meio do comando import médicos, dessa forma, assim que necessário teremos acesso aos dados contidos em nosso arquivo médicos.py.

Em seguida criamos uma variável de nome menu que pede ao usuário que digite S ou N caso queira ou não agendar uma consulta. Aqui realizamos duas validações, a primeira convertendo o que o usuário digitar para tipo string, e a segunda convertendo o que o usuário digitar para maiúsculo.

Na sequência criamos uma estrutura condicional, onde inicialmente caso o valor atribuído para menu seja igual a 'S', o usuário entrará na fase de agendamento da consulta, caso essa condição inicial não seja verdadeira, é exibido em tela apenas uma mensagem agradecendo o contato...

Em nossa estrutura condicional, validade como 'S', inicialmente temos uma variável de nome paciente que via input( ) pede que o usuário digite seu nome, em seguida criamos algumas funções print( ) exibindo em tela um simples menu para que o usuário escolha com qual médico queira consultar.

Dando continuidade, declaramos uma variável de nome medico que recebe do usuário um dado/valor correspondente a qual médico o mesmo escolheu anteriormente.

Aqui criamos uma nova estrutura condicional dentro da anterior, onde caso o que o usuário tenha digitado para variável medico seja igual a 1, instanciamos de nosso módulo medicos a variável medicos que por sua vez, como lista, em sua posição de índice 0, fará referência a Dr.a 'Grazielle Veiga'. Caso o número digitado pelo usuário tenha sido 2, usamos do mesmo processo porém instanciando de medicos o elemento de índice 1 da lista.

Supondo que o usuário tenha digitado N ou qualquer outro caractere, o retorno será:
Agradecemos seu contato!!!

Supondo que o usuário tenha digitado S em seguida seu nome e a opção 1 do menu, o retorno será:
Fernando, escolha com qual médico deseja consultar:
1 – Grazielle Veiga
2 – Matheus Correa
Sua consulta com a Dr.a Grazielle Veiga será agendada.

---

63 - Aprimore o exemplo anterior, incluindo um módulo simulando o cadastro de usuários em um plano de saúde, apenas permitindo o agendamento de consulta caso o usuário que está interagindo com o programa conste no cadastro:

```
# Arquivo medicos.py
medicos = ['Grazielle Veiga',
           'Matheus Correa']

# Arquivo cadastro_plano_saude.py
usuarios = {'001':'Fernando Feltrin',
            '002':'Ana Clara'}

# Arquivo main.py
import medicos
import cadastro_plano_saude

usuario = str(input('Digite seu número de usuário: '))
if usuario in cadastro_plano_saude.usuarios.keys():
  if usuario == '001':
    usuario = 'Fernando'
    print('Bem-vindo Fernando!!!')
    #return usuario
  if usuario == '002':
    usuario = 'Ana Clara'
    print('Bem-vinda Ana Clara!!!')
    #return usuario
  else:
    print('Usuário desconhecido ou não cadastrado.')

  menu = str(input('Deseja agendar uma consulta? (S ou N) ')).upper()
  if menu == 'S':
    print('Escolha com qual médico deseja consultar:')
    #print(f'{usuario}, escolha com qual médico deseja consultar:')
    #caso tenha retornado o nome do usuário na condicional anterior
    print('1 - Grazielle Veiga')
    print('2 - Matheus Correa')
    medico = int(input('Com qual médico deseja agendar consulta?'))
    if medico == 1:
      print(f'{usuario}, sua consulta com a Dr.a {medicos.medicos[0]} está agendada.
')
    if medico == 2:
      print(f'{usuario}, sua consulta com o Dr. {medicos.medicos[1]} está agendada.')
  else:
    print('Agradecemos o seu contato!!!')
```

```
else:
  print('Usuário não cadastrado.')
```

Obedecendo o enunciado da questão, faremos algumas alterações tornando nosso programa um pouco mais robusto.

Inicialmente criamos um novo módulo, sendo este o arquivo cadastro_plano_saude.py, onde dentro do mesmo consta a variável usuarios por sua vez parametrizada com um dicionário, onde chaves e valores correspondem ao dígito identificador e ao nome do usuário, respectivamente.

Segunda alteração, ao iniciar a escrita do código de nosso arquivo main.py devemos realizar as importações tanto do módulo medicos quanto do módulo cadastro_plano_saude. Lembrando que estes módulos ao serem importados não precisam da extensão .py, porém os mesmos devem estar no mesmo diretório de nosso arquivo main.py.

Na sequência implementamos a estrutura de validação que irá identificar o usuário para fornecer ao mesmo a opção de agendamento de consulta. Sendo assim, criamos uma variável de nome usuário que por meio da função input( ) pede que o usuário digite seu dígito identificador.

Caso o número fornecido e atribuído a usuario conste nas chaves do dicionário usuarios do módulo cadastro_plano_saude, então é realizada a identificação do usuário, caso contrário, é exibida uma mensagem de erro dizendo que o usuário é desconhecido ou não está cadastrado na base de dados.

De resto todo o restante do código permanece igual ao exemplo anterior, apenas com a diferença que agora todo esse bloco é indentado à estrutura condicional de validação de usuário.

Supondo que o usuário tenha digitado 003, o retorno será:
Usuário desconhecido ou não cadastrado.

Supondo que o usuário tenha digitado 002, seguido de S para agendar a consulta de 2 para escolher o segundo médico da lista, o retorno será:
Bem-vinda Ana Clara!!!
Escolha com qual médico deseja consultar:
1 – Grazielle Veiga
2 – Matheus Correa
Ana Clara, sua consulta com o Dr. Matheus Correa está agendada.

---

64 - Crie uma função que recebe parâmetros tanto por justaposição quanto nomeados a partir de uma lista e de um dicionário, desempacotando os elementos e reorganizando os mesmos como parâmetros da função:

```
numeros = (33, 1987, 2020)
dados = {'Nome':'Fernando', 'Profissão':'Engenheiro'}

def identificacao(*args, **kwargs):
  print(args)
  print(kwargs)

identificacao(*numeros, **dados)
```

*Aqui temos um exercício com uma proposta um pouco diferente dos exemplos anteriores onde para *args tínhamos apenas um dado/valor. Em Python é perfeitamente possível passar mais de um dado/valor que ocupará por justaposição campos de parâmetros referentes a *args, desde que os dados/valores estejam em formado de set/tupla ou conjunto. Já para os dados em **kwargs sabemos que estaremos repassando dados/valores como parâmetros nomeados, ou seja, com valores padrão pré-definidos.*

Para nosso exemplo temos uma variável de nome numeros que atribuída para si tem uma tupla com alguns números, da mesma forma demos uma variável de nome dados que como atributo tem um dicionário com alguns dados organizados em chaves e valores.

*Na sequência criamos nossa função identificacao( ) que receberá *args e **kwargs, exibindo em tela tais dados.*

*Por fim chamamos nossa função identificacao( ) repassando como parâmetros em *args para a mesma os dados da variável numeros, para isso instanciamos *numeros com o marcador “ * “ referente a *args. O mesmo processo é feito, repassando o conteúdo de dados como **dados, assim ocupando os campos em **kwargs.*

O retorno será:
(33, 1987, 2020)
{‘Nome’ : ‘Fernando’, ‘Profissão’ : ‘Engenheiro’}

---

65 - Crie uma classe de nome Carro e lhe dê três atributos: nome, ano e cor.

```
class Carro:
  def __init__(self, nome, ano, cor):
    self.nome = nome
    self.ano = ano
    self.cor = cor
```

Retomando o básico de orientação a objetos, pela sintaxe uma classe deve ser identificada pela palavra reservada class e por convenção costumamos dar um nome para a classe iniciado com letra maiúscula.

Sendo assim, declaramos nossa classe Carro, dentro do corpo dessa classe criamos o método construtor def __init__( ) por sua vez com os atributos de classe nome, ano e cor, além da palavra reservada self, referente ao escopo da classe. Lembrando que uma classe simples pode ou não ter um método construtor.

Indentado ao bloco do método construtor criamos os objetos self.nome, self.ano e self.cor que receberão dados quando repassados como atributos de classe.

---

66 - Crie uma classe Pessoa, instancie a mesma por meio de uma variável e crie alguns atributos de classe dando características a essa pessoa. Por fim exiba em tela alguma mensagem que incorpore os atributos de classe criados:

```
class Pessoa:
  pass

pessoa1 = Pessoa()
pessoa1.nome = 'Fernando'
pessoa1.idade = 33
pessoa1.profissao = 'Professor'

print(f'{pessoa1.nome}, {pessoa1.idade} é um {pessoa1.profissao}...')
```

Inicialmente criamos nossa classe Pessoa, inicialmente vazia.

Em seguida declaramos uma variável de nome pessoa1 que instancia a classe Pessoa( ), criando os atributos de classe nome, idade e profissao com seus respectivos dados.

Por fim via função print( ) e f'strings criamos uma mensagem onde nas máscaras de substituição instanciamos os objetos criados anteriormente.

O retorno será:
Fernando, 33, é um Professor...

---

67 - Crie uma classe que armazena algumas características de um carro, em seguida crie dois carros distintos, de características diferentes, usando da classe para construção de seus objetos/variáveis.

```
class Carro:
  ano = 2020
  cor = 'prata'
  modelo = 'hatch'
  opcionais = 'nenhum'

carro1 = Carro()

carro2 = Carro()
carro2.ano = 2016
carro2.cor = 'preto'
carro2.modelo = 'sedan'
```

```
print(f'Carro nº1: Ano = {carro1.ano}, Cor = {carro1.cor}, Modelo = {carro1.modelo}
, Opcionais = {carro1.opcionais}')

print(f'Carro nº2: Ano = {carro2.ano}, Cor = {carro2.cor}, Modelo = {carro2.modelo}
, Opcionais = {carro2.opcionais}')
```

Criada nossa classe Carro, dentro da mesma criamos alguns objetos simples ano, cor, modelo e opcionais com dados/valores padrão 2020, 'prata', 'hatch' e 'nenhum', respectivamente.

Na sequência criamos uma variável de nome carro1, que por sua vez instancia a classe Carro( ), automaticamente herdando suas características de seus atributos de classe.

De forma parecida criamos uma variável carro2, porém essa não somente instancia a classe Carro( ) mas modifica alguns dos atributos de classe para gerar um veículo diferente de carro1.

Por fim, exibimos em tela via função print( ) as características de ambos os carros fazendo uso de f'strings inserindo nas máscaras de substituição os dados/valores dos atributos de classe referentes a cada veículo.

O retorno será:

Carro nº1: Ano = 2020, Cor = prata, Modelo = hatch, Opcionais = nenhum
Carro nº2: Ano = 2016, Cor = preto, Modelo = sedan, Opcionais = nenhum

---

68 - Crie uma classe Pessoa com método inicializador e alguns objetos de classe vazios dentro da mesma que representem características de uma pessoa:

```
class Pessoa:
  def __init__(self):
    self.nome = None
    self.idade = None
    self.altura = None
    self.peso = None
    self.sexo = None
```

Atendendo o enunciado da questão, criamos uma classe de nome Pessoa, dentro de sua estrutura criamos o método inicializador/construtor def __init__(self), assim como os objetos self.nome, self.idade, self.altura, self.peso, self.sexo, todos definidos como None.

Aqui temos dois comportamentos a ser entender, primeiro deles que como atributos de classe não foi definido nada além de self, sendo assim, ao instanciar essa classe não temos a obrigatoriedade de repassar "parâmetros" para a mesma, ainda assim todos seus objetos internos são perfeitamente instanciáveis.

Outro ponto a se destacar é que não podemos criar variáveis/objetos vazios em Python (no sentido de não lhes passar algum atributo no momento da declaração, porém, podemos atribuir para esses objetos a palavra reservada None, que identificará para o interpretador que aquele objeto inicialmente não possui nenhum dado/valor, mas que mesmo assim sua estrutura é iterável.

---

69 - Crie uma classe Inventario com os atributos de classe pré-definidos item1 e item2, a serem cadastrados manualmente pelo usuário, simulando um simples carrinho de compras:

```
class Inventario:
  def __init__(self, item1, item2):
    self.item1 = item1
    self.item2 = item2
```

```
cliente1 = Inventario('Camisa Adidas Tam GG', 'Calça Jeans Tam 50')

print(cliente1.item1)
print(cliente1.item2)
```

De forma parecida com alguns exercícios anteriores, aqui criaremos uma classe que servirá como um molde para criação de outros objetos. Como esses objetos devem poder ser acessados tanto de dentro quanto de fora da classe, é interessante que criemos para a classe um método inicializador/construtor.

Sendo assim, criamos nossa classe Inventario com seu método inicializador __init__(self), além é claro dos atributos de classe item1 e item2 conforme o enunciado.

Também criamos os objetos self.item1 e self.item2 que receberão os dados/valores dos atributos de classe.

Na sequência criamos uma variável de nome cliente1, que instancia nossa classe Inventario( ) "parametrizando" a mesma com 'Camisa Adidas Tam GG' e 'Calça Jeans Tam 50', respectivamente.

Por meio de nossa função print( ) é possível visualizar se de fato tais objetos foram criados e associados a variável cliente1.

O retorno será:
Camisa Adidas Tam GG
Calça Jeans Tam 50

---

70 - Crie uma classe Biblioteca que possui uma estrutura molde básica para cadastro de um livro de acordo com seu título, porém que espera a inclusão de um número não definido de títulos. Em seguida cadastre ao menos 5 livros nessa biblioteca:

```
class Biblioteca:
  def __init__(self, livro1, **kwargs):
    self.livro1 = livro1
```

```
prateleira1 = Biblioteca('LIVRO TESTE')
prateleira1.livro2 = '1984 - George Orwell'
prateleira1.livro3 = 'Duna - Frank Herbert'
prateleira1.livro4 = 'O Iluminado - Stephen King'
prateleira1.livro5 = 'O Exorcista - William Peter Blatty'
prateleira1.livro6 = 'O Hobbit - J. R. R. Tolkien'

print(prateleira1.livro4)
print(prateleira1.livro1)
```

*Criada a classe Biblioteca, em seu método inicializador/construtor temos uma particularidade, como não sabemos ao certo quantos livros irão compor essa biblioteca, uma vez que seu acervo será atualizado ao longo do tempo e não todo inicialmente, podemos usar de **kwargs como atributo de classe.*

*Como visto anteriormente em exercícios de funções, sempre que precisarmos usar de múltiplos parâmetros, porém não temos seu número fixo definido inicialmente, podemos reservar o espaço para tais parâmetros via *args e **kwargs. Em nosso exemplo estaremos criando objetos os quais possuem como atributo um título de um livro, estrutura muito parecida com a de um parâmetro nomeado.*

*Dessa forma, usando **kwargs como atributo de classe podemos perfeitamente repassar para esse campo o atributo com seu respectivo dado/valor tendo em vista que o mesmo será reconhecido como chave e valor de um dicionário.*

*Abstraindo essa ideia em código, no método inicializador de nossa classe Biblioteca criamos um atributo de classe livro1 que associará seu dado para o objeto self.livro1 = livro1, para todos os demais, à medida que formos criando novos objetos os mesmos estarão ocupando os campos de **kwargs.*

Já fora do escopo da classe Biblioteca, criamos uma variável de nome prateleira1 que instancia e inicializa nossa classe Biblioteca( ) repassando para a mesma 'LIVRO TESTE' que por sua vez alimentará o campo de livro1.

*Em seguida, ao aplicar o método prateleira1.livro2, estaremos criando o objeto self.livro2 e o mesmo estará indexado para **kwargs. O processo é repetido para criação de 5 livros diferentes conforme o enunciado.*

Por fim, para verificar se a lógica estava correta, por meio de nossa função print( ) por sua vez parametrizada com alguns dos livros cadastrados, podemos verificar o conteúdo destes objetos.

O retorno será:
LIVRO TESTE
O Iluminado – Stephen King

---

71 - Crie uma calculadora simples de 4 operações (soma, subtração, multiplicação e divisão) usando apenas de estrutura de código orientada a objetos:

```
class Calculadora:

  def soma(self, n1, n2):
    return n1 + n2

  def subtracao(self, n1, n2):
    return n1 - n2

  def multiplicacao(self, n1, n2):
    return n1 * n2

  def divisao(self, n1, n2):
    return n1 / n2

n1 = Calculadora()

n1.multiplicacao(5, 4)
```

Criada a classe Calculadora, podemos criar métodos de classe, um para cada operação matemática, de forma que a variável/objeto que instanciar essa classe pode individualmente usar da função matemática a qual se faz necessária.

Para operação de soma, por exemplo, definimos soma( ) que recebe dois números n1 e n2, retornando para a classe a soma dos conteúdos desses parâmetros.

O mesmo é feito para cada operação, retornando apenas o valor da função matemática da mesma.

No corpo geral de nosso código criamos uma variável de nome n1 que instancia e inicializa nossa classe Calculadora( ), em seguida usando do método de classe necessário, parametrizando o mesmo com os valores a serem considerados na operação matemática.

O retorno será:
20

---

72 - Mostre via terminal a string ‘Bem vindo ao mundo da programação!!!’ de trás para frente usando indexação:

```
variavel = 'Bem vindo ao mundo da programação!!!'

print(variavel[::-1])
```

Declarada a variável variavel1 que por sua vez possui como atributo a string ‘Bem Vindo ao mundo da programação!!!, para exibir em tela essa string de trás para frente basta que para nossa variável usemos da notação [::-1], uma vez que desse modo estamos lendo todos os caracteres da string, porém realizando a leitura de cada elemento do último para o primeiro.

O retorno será:
!!!oãçamargorp ad odnum ao odniv meB

---

73 - Escreva um programa que encontre todos os números que são divisíveis por 7, mas que não são múltiplos de 5, entre 2000 e 2200 (ambos incluídos). Os números obtidos devem ser impressos em sequência, separados por vírgula e em uma única linha:

```
lista=[]
for i in range(2000, 2201):
    if (i%7==0) and (i%5!=0):
        lista.append(str(i))

print(','.join(lista))
```

Aqui temos um daqueles exercícios clássicos onde no enunciado são exigidas várias coisas a serem realizadas, necessitando que desmembremos essas instruções em partes para poder entender a lógica a ser utilizada no exercício.

Inicialmente criamos uma variável lista que inicialmente é uma lista vazia onde iremos inserir os números divisíveis por 7 que não são múltiplos de 5.

Em seguida criamos um laço for para poder percorrer o intervalo numérico definido na questão, nesse caso entre 2000 e 2200, retornando o valor de cada leitura a variável temporária i. Lembrando que ao definir o último número do intervalo devemos acrescentar 1 ao seu valor, pois o último elemento lido na verdade servirá de gatilho para que o laço interrompa seu processo.

Dentro desse laço criamos a estrutura lógica/condicional que resolve o problema proposto na questão. Aqui definimos que se o módulo da divisão de i por 7 for igual a 0 e o módulo da divisão de 5 for diferente de 0, esse número será inserido na lista via método append( ).

Repare que aqui temos uma estrutura condicional de lógica composta, em função de usar do operador lógico " and ", as duas proposições devem ser verdadeiras para que o número em questão atenda a condição que lhe é imposta.

Por fim, via função print( ) exibimos em tela os valores que foram inseridos em nossa variável lista, separados por vírgula conforme o exigido pelo enunciado da questão.

O retorno será:
2002, 2009, 2016, 2023, 2037, 2044, 2051, 2058, 2072, 2079, 2086, 2093, 2107, 2114, 2121, 2128, 2141, 2149, 2156, 2163, 2177, 2184, 2191, 2198

---

74 - Escreva um programa, uma calculadora simples de 4 operações, onde o usuário escolherá entre uma das 4 operações (soma, subtração, multiplicação e divisão). Lembrando que o usuário digitará apenas dois valores, e escolherá apenas uma operação matemática do menu.

```
print('CALCULADORA DE 4 OPERAÇÕES')
print('+ -> Soma')
print('- -> Subtração')
print('* -> Multiplicação')
print('/ -> Divisão')
a = float(input("Digite o primeiro número:"))
b = float(input("Digite o segundo número:"))
operacao = input("Digite a operação a realizar (+, -, * ou /):")

if operacao == "+":
 soma = a + b
 print(f'O resultado da soma é: {soma}')
elif operacao == "-":
 subtracao = a - b
 print(f'O resultado da subtração é: {subtracao}')
elif operacao == "*":
 multiplicacao = a * b
 print(f'O resultado da multiplicação é: {multiplicacao}')
elif operacao == "/":
 divisao = a / b
 print(f'O resultado da divisão é: {divisao}')
else:
 print("Operação inválida!")
```

De acordo com o enunciado da questão, inicialmente criamos um menu de opções a ser mostrado para o usuário, seguido da interação com o mesmo para que dê entrada nos valores e escolha a operação a qual realizar com tais números.

O que diferencia esse exercício de outros propostos para uma "calculadora" é o uso de estruturas condicionais onde apenas um desfecho é permitido.

No início de nossa estrutura condicional simplesmente estamos verificando se o dado/valor de nossa variável operacao é igual a "+", caso essa condição seja verdadeira é realizada a soma

dos valores atribuídos às variáveis a e b, exibindo em tela o resultado dessa soma.

Todo o resto é feito da mesma forma, apenas alterando a operação matemática em si e a mensagem de retorno, porém fazendo o uso de elif para que apenas uma dessas condições seja considerada verdadeira, de modo que quando o usuário selecionar uma, todas as outras serão descartadas, e ao término do processo da operação escolhida todo o programa é encerrado.

O retorno será:
CALCULADORA DE 4 OPERAÇÕES
+ -> Soma
- -> Subtração
* -> Multiplicação
/ -> Divisão
Digite o primeiro número: 6
Digite o segundo número: 8
Digite a operação a realizar (+, -, * ou /)
O resultado da multiplicação é 48.0

---

75 - Crie uma função que recebe um número e retorna o mesmo dividido em duas metades, sendo cada metade um elemento de uma lista:

```
def divide_pela_metade(n):
  return [n//2, n - n//2]

n = int(input('Digite um número: '))
print(divide_pela_metade(n))
```

Aqui temos um problema simples de resolver desde que entendamos a lógica a ser implementada. Note que o que é pedido na questão é que não somente o número seja dividido "ao meio", mas que cada parte seja um elemento de uma lista.

Sendo assim, criamos nossa função divide_pela_metade( ) que recebe como parâmetro um número. Dentro do corpo dessa função temos um retorno, onde já criamos a estrutura de uma lista, sendo na posição do primeiro elemento a operação de dividir o mesmo por 2, assim como na posição do segundo elemento estaremos subtraindo o valor do número pela divisão dele mesmo por 2.

Assim teremos em cada posição dessa lista um elemento representando a metade do número repassado anteriormente.

Por fim, criamos a estrutura que via input( ) interage com o usuário pedindo que o mesmo insira um número, atribuindo esse valor a variável n.

Finalizando, diretamente em nossa função print( ) podemos chamar a função divide_pela_metade( ) parametrizando a mesma com o valor de n.

Supondo que o usuário tenha digitado 26, o retorno será:
[13, 13]

---

76 - Crie um programa que gera um dicionário a partir de um valor digitado pelo usuário, de modo que serão exibidos todos valores antecessores a este número multiplicados por eles mesmos. Supondo que o usuário tenha digitado 4, a saída deve ser {1: 1, 2: 4, 3: 9, 4: 16} :

```
numero = int(input('Digite um número: '))
dicionario = dict()

for i in range(1, numero + 1):
    dicionario[i] = i * i

print(dicionario)
```

De acordo com o exercício proposto, inicialmente criamos uma variável de nome numero que recebe do usuário um número. Também criamos uma variável de nome dicionario que atribuída para si tem um gerador de dicionário, inicialmente sem dados compondo o mesmo.

Na sequência por meio de um laço for percorreremos todos elementos iniciando em 1 até chegar no número digitado pelo usuário. Lembrando de validar o número acrescentando 1 unidade ao mesmo como gatilho de encerramento do laço for.

Dentro do corpo do laço for instanciamos nosso dicionario, agora a cada laço iterando com uma posição do mesmo, inserindo nesse campo o valor de i multiplicado por ele mesmo.

Por fim, por meio da função print( ) exibimos em tela o conteúdo de nosso dicionário.

Supondo que o usuário tenha digitado 14, o retorno será:
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196}

---

77 - Defina uma função que pode aceitar duas strings como entrada, exibindo em tela apenas a string de maior tamanho/comprimeito. Caso as duas strings tenham mesmo tamanho, exiba em tela as duas:

```
def mostra_msg_maior(s1, s2):
    len1 = len(s1)
    len2 = len(s2)
    if len1 > len2:
        print(f'{s1} é a string de maior tamanho.')
    elif len2 > len1:
        print(f'{s2} é a string de maior tamanho.')
    else:
        print(f'{s1} e {s2} são strings de mesmo tamanho.')
        print(s1)
        print(s2)
```

Pelo enunciado da questão é possível ver de imediato que aqui temos um problema de lógica simples, bastando comparar as strings e instanciar a maior delas em uma mensagem de retorno.

Sendo assim, inicialmente criamos nossa função mostra_msg_maior( ) que recebe s1 e s2 como parâmetros. Já no corpo da função criamos as variáveis len1 e len2 que por meio do método len( ) verifica o tamanho das strings repassadas para s1 e s2.

Em seguida criamos a estrutura condicional em si, o de definimos que se o tamanho de len1 for maior que o tamanho de len2, é exibida em tela o conteúdo da variável s1.

Da mesma forma, caso o tamanho de len2 seja maior que o tamanho de len1, o conteúdo de s2 que é exibido em tela via função print( ).

Por fim, caso nenhuma das duas condições anteriores seja verdadeira, subentende-se então que as strings em s1 e s2 tem o mesmo tamanho, então exibimos as duas em tela.

No corpo geral de nosso código chamamos nossa função mostra_msg_maior( ) parametrizando a mesma com duas strings “Fernando” e “Python”.

O retorno será:
Fernando é a string de maior tamanho.

---

78 - Escreva um programa que recebe um texto do usuário e o converte para código Morse, exibindo em tela o texto em formato Morse, seguindo a padronização “ . - " (ponto, traço).

```
def texto_p_morse(texto):
    d = {'A': '.-', 'B': '-...',
         'C': '-.-.', 'D': '-..', 'E': '.',
         'F': '..-.', 'G': '--.', 'H': '....',
         'I': '..', 'J': '.---', 'K': '-.-',
         'L': '.-..', 'M': '--', 'N': '-.',
         'O': '---', 'P': '.--.', 'Q': '--.-',
         'R': '.-.', 'S': '...', 'T': '-',
         'U': '..-', 'V': '...-', 'W': '.--',
         'X': '-..-', 'Y': '-.--', 'Z': '--..',
         '1': '.----', '2': '..---', '3': '...--',
         '4': '....-', '5': '.....', '6': '-....',
         '7': '--...', '8': '---..', '9': '----.',
         '0': '-----', ',': '--..--', '.': '.-.-.-',
         '?': '..--..', '/': '-..-.', '-': '-....-',
         '(': '-.--.', ')': '-.--.-', '!': '-.-.--',
         ' ': ' ', '"': '.----.', ':': '---...'}

    return ' '.join(d[i] for i in texto.upper())

texto = input('Digite o texto a ser convertido para Código Morse: ')
conversor = texto_p_morse(texto)

print(f'{texto} em Código Morse: {conversor}')
```

O primeiro ponto a ser abordado nesse exercício é o fato de que para se converter um determinado texto para código Morse devemos buscar um dicionário que equipare caracteres normais em padrão Morse (ponto, traço), o que é facilmente encontrado com uma rápida pesquisa na internet.

Dessa forma, criamos nossa função texto_p_morse( ) que recebe um texto como parâmetro. Dentro da estrutura de código dessa função criamos nosso dicionário tradutor de Morse.

Como retorno da função em si, bastaria simplesmente associar chave e valor do dicionário, dando entrada de um caractere e recebendo seu código Morse correspondente, porém, é interessante aqui realizarmos algumas validações.

Primeira delas eliminando todo e qualquer espaço da string fornecida como dado de entrada, uma vez que para Morse não há necessidade de interpretar os espaços, assim como todo e qualquer texto inserido é convertido para maiúsculo, para ser identificado corretamente quando associado ao dicionário.

Validações feitas, agora sim podemos criar um laço for que percorre cada elemento do texto fornecido pelo usuário, retornando do dicionário seu código equivalente.

Já no escopo global de nosso código criamos uma variável de nome texto que via input( ) recebe um texto do usuário, de forma parecida criamos a variável conversor que chama nossa função texto_p_morse( ) parametrizando a mesma com o conteúdo da variável texto.

Por fim, via função print( ), fazendo uso de f'strings criamos uma mensagem interpolando em suas máscaras de substituição o texto original e seu código Morse.

Supondo que o usuário tenha digitado 'Python', o retorno será:

Python em Código Morse: **.--. -.-- - .... --- -.**

---

79 - Escreva um programa que recebe do usuário um número de 0 a 100 e transcreve o mesmo por extenso. Por exemplo o usuário digita 49, então é retornara a string ‘quarenta e nove’.

```
def num_p_texto(n):
    if n == 0:
        return 'zero'

    unidade = ('', 'um', 'dois', 'tres', 'quatro', 'cinco','seis', 'sete', 'oito', 'nove')

    dezena = ('', '', 'vinte e', 'trinta e', 'quarenta e', 'cinquenta e', 'sessenta e', 'setent
a e', 'oitenta e', 'noventa e')

    dez_x = ('dez', 'onze', 'doze', 'treze', 'quatorze', 'quinze', 'dezesseis', 'dezessete',
 'dezoito', 'dezenove')

    h, t, u = '', '', ''

    if n == 100:
        h = unidade[0] + 'cem'
        n = n % 100
    elif n >= 20:
        t = dezena[n // 10]
        n = n % 10
    elif n >= 10:
        t = dez_x[n - 10]
        n = 0

    u = unidade[n]
    return ' '.join(filter(None, [h, t, u]))

num = int(input('Digite o número a ser convertido: '))

if num <= 100:
  print(f'{num} por extenso é: {num_p_texto(num)}')
else:
  print('Número Inválido!!!')


if num <= 100:
  print(f'{num} por extenso é: {num_p_texto(num)}')
else:
  print('Número Inválido!!!')
```

Para esse exercício podemos usar de uma abordagem um pouco parecida com a do exercício onde convertemos um certo texto para Morse, porém aqui, temos que levar em consideração que quando estamos escrevendo por extenso um determinado número, existe uma nomenclatura específica para unidades, dezenas, centenas (que não usaremos de acordo com o enunciado do exercício) e dos números intermediários entre 10 a 19.

Sendo assim, também estaremos fazendo a equivalência de um determinado número com um elemento de uma lista via índice para substituir número por texto, mas com algumas diferenças se comparado ao exercício anterior.

Inicialmente criamos nossa função num_p_texto( ) que receberá um número a ser convertido. Dentro do corpo dessa função inicialmente criamos uma estrutura condicional onde se o valor de n for igual a 0, é retornado apenas 'zero', mas caso essa condição não seja verdadeira, aí sim partimos para as conversões número -> texto.

Então criamos uma variável de nome unidade, com números descritos por extenso desde nenhum, 'um' até 'nove'. Como uma espécie de validação, vamos criar essas estruturas em formato de tupla para que não sejam modificadas.

Também criamos uma tupla atribuída a dezena com números descritos desde 'vinte e' até 'noventa e'. O mesmo é feito para a variável dez_x, onde os elementos da tupla correspondem desde 'dez' até 'dezenove'.

Finalizando essa parte criamos as variáveis h, t e u, inicialmente com strings vazias. Caso você não tenha notado, o grande diferencial aqui, quando comparado ao exercício anterior é que para montar a nomenclatura de um determinado número por extenso, o mesmo terá de percorrer nossas variáveis unidade, dezena e dez_x para montar os blocos que formam o nome do número.

Finalizada a estrutura anterior, criamos agora nossa segunda estrutura condicional, ainda indentada como a anterior, verificando se o valor de n é igual a 100, caso sim, para retornar apenas 'cem' temos de instanciar nossa variável h, atribuindo para a mesma o conteúdo de unidade em sua posição de índice 0 concatenado com a string 'cem'. Por fim, n tem seu valor atualizado com o módulo da divisão de seu valor por 100.

De forma parecida, em nosso elif definimos como condição que se o valor de n for igual ou maior a 20, nossa variável t recebe como atributo o conteúdo de dezena, na posição de índice que corresponde a divisão exata do número por 10. Aqui, n tem seu valor atualizado para o módulo da divisão de seu valor por 10.

E finalmente em nossa terceira estrutura condicional, caso nenhuma das anteriores seja verdadeira, estipulamos que se o valor de n foi igual ou maior que 10, a variável t recebe como atributo o conteúdo de dez_x em posição de índice n subtraído de 10. Finalizando essa etapa, n tem seu valor atualizado simplesmente para 0.

É criada uma última variável dentro da função de nome u, que recebe unidade[n], em outras palavras, recebe o último valor atribuído para n.

Encerrando a função em si, para que possamos concatenar os nomes associados ao número lido simplesmente via método filter( ) juntamos as strings que formaram os últimos dados atribuídos para as variáveis h, t e u, juntando as mesmas com um espaço entre elas.

Em seguida, agora trabalhando fora da função, criamos uma variável de nome num, que via input( ) lê um número a ser convertido, digitado pelo usuário.

Criamos uma última estrutura condicional pois de acordo com o enunciado dessa questão, os números lidos não podem ser maiores que 100, então simplesmente definimos que se o valor de num for igual ou menor que 100, é exibida em tela uma mensagem

onde via f'strings interpolamos o valor de num, assim como na segunda máscara de substituição chamamos nossa função num_p_texto( ) parametrizando a mesma com o valor de num.

Caso essa condição não seja válida, simplesmente é retornado ao usuário uma mensagem de número inválido pois subentende-se que o mesmo tenha digitado um número maior que 100, algo que não esperamos em nossa função.

Supondo que o usuário tenha digitado 76, o retorno será:
76 por extenso é: setenta e seis

---

80 - Crie uma função que recebe um nome e um sobrenome do usuário, retornando os mesmos no padrão americano, ou seja, sobrenome primeiro, seguido do nome:

```
def inverte_nome(n):
  nome, sobrenome = n.split()
  return ' '.join([sobrenome, nome])
```

```
pessoa = input('Digite seu nome e sobrenome: ')
pessoa = inverte_nome(pessoa)

print(pessoa)
```

Aqui temos mais um exercício que pode ser resolvido de múltiplas formas, porém, para resolver o que é pedido pela questão, vamos nos ater ao meio considerado mais fácil.

Inicialmente criamos nossa função inverte_nome( ) que receberá um nome a ser tratado. Em seguida, dentro do corpo da função, declaramos duas variáveis nome e sobrenome, que via método split( ) dividirá o conteúdo de n atribuindo cada metade a sua respectiva variável.

Encerrando a função, é retornado ao usuário por meio do método join( ) os dados de sobrenome e nome, nesta ordem, separando cada palavra com um espaço.

Fora da função, criamos uma variável de nome pessoa que via input( ) pede que o usuário digite seu nome e sobrenome.

Na sequência, a variável pessoa chama a função inverte_nome( ) parametrizando a mesma com seu próprio conteúdo, atualizando assim seu conteúdo.

Por fim é exibido em tela o conteúdo da variável pessoa.

Supondo que o usuário tenha digitado Fernando Feltrin, o retorno será:
Feltrin Fernando

---

81 - Crie um programa que gera uma senha aleatória, com um tamanho definido pelo usuário:

```
import random

tamanho = int(input("enter the length of password"))

s = "abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVW
XYZ!@#$%^&*()?"

senha =  "".join(random.sample(s, tamanho))

print(senha)
```

Para implementar certas funcionalidades aos nossos códigos, uma prática comum é buscar alguma biblioteca, módulo ou pacote que já disponha de tais funcionalidades, não sendo necessário desenvolver tudo do zero.

Em nosso exercício atual, onde temos um problema o qual se faz necessário a geração de números aleatórios, podemos perfeitamente realizar essa ação por meio de uma função já pronta da biblioteca random.

Inicialmente importamos a biblioteca random por meio do comando import random.

Na sequência criamos uma variável de nome tamanho, que recebe validado como número inteiro um número fornecido pelo usuário para ser usado como parâmetro para o tamanho da senha a ser gerada.

Criamos também uma variável de nome s que por sua vez tem uma string com todos os possíveis caracteres a serem usados para a senha gerada.

Na próxima etapa criamos uma variável de nome senha que por sua vez recebe como atributo o retorno do método random.sample( ), parametrizado com o conteúdo de s e com o tamanho definido em tamanho.

Para finalizar, é exibido em tela a senha gerada por meio da função print( ), parametrizada com o conteúdo de senha.

Em meu caso, ao digitar 14 quando questionado, o retorno foi:
sfC^1b(4?8rqDh

*Importante salientar que fazendo o uso da biblioteca random, a cada execução será gerada uma senha diferente, como é esperado.

---

82 - Crie uma função que exibe em tela tanto o conteúdo de uma variável local quanto de uma variável global, sendo as variáveis de mesmo nome, porém uma não substituindo a outra:

```
def msg():
    global mensagem
    print(mensagem)
    mensagem = 'Usufrua das funcionalidades do sistema.'
    print(mensagem)

mensagem = 'Olá, seja bem-vindo(a)!'
msg()
```

Para esse exercício, de acordo com o que é pedido em seu enunciado, usaremos de duas variáveis de mesmo nome, mas propósitos e escopos diferentes. Logo, para que não haja conflito entre as mesmas ou que essas variáveis não se sobreponham/atualizem, devemos usar de algum marcador que as separe quanto seu escopo.

Inicialmente criamos nossa função msg( ), sem parâmetros mesmo. Para instanciarmos a variável mensagem do escopo global do código (de fora do corpo da função), devemos usar o marcador global antes do nome da variável. Em seguida simplesmente exibimos em tela seu conteúdo via função print( ).

Na sequência, é criada outra variável de nome mensagem, esta sem marcador nenhum, fará parte do escopo local da função. Também exibimos em tela seu conteúdo via print( ).

Pela própria leitura léxica, na primeira linha do corpo de nossa função msg( )é lida e instanciada a variável mensagem do escopo global, graças ao seu marcador global, na quarta linha do corpo da função quando instanciamos uma variável de nome mensagem, essa terá o comportamento de outra variável, dessa vez local.

O retorno será:
Olá, seja bem-vindo(a)!
Usufrua das funcionalidades do sistema.

---

83 – Crie um programa que recebe um número digitado pelo usuário, convertendo o mesmo para algarismo de número Romano, exibindo em tela esse dado já convertido:

```
class Conversor:
    def int_para_romano(self, num):
        val_int = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        val_rom = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        numero_romano = ''
        i = 0
        while  num > 0:
            for _ in range(num // val_int[i]):
                numero_romano += val_rom[i]
                num -= val_int[i]
            i += 1
        return numero_romano
```

```
num = int(input('Digite um número a ser convertido: '))

print(Conversor().int_para_romano(num))
```

De modo parecido com o feito nos exercícios de conversão de número para Morse e conversão de algarismo para número por extenso, aqui temos uma aplicação interessante baseada em conversão onde iremos equiparar um número convencional para sua forma escrita em número Romano.

Inicialmente criamos nossa classe Conversor, dentro da mesma o método de classe int_para_romano( ), trabalhando dentro do escopo da classe (self), recebendo um atributo de classe, nesse caso, um número.

Dentro do corpo deste método de classe criamos uma lista de nome val_int, que por sua vez possui toda cadeia de algarismos necessários para que possamos representar qualquer número.

Da mesma forma criamos uma segunda lista, essa de nome val_rom, com os algarismos em sua representação romana, de forma que possa cobrir o mesmo número de algarismos em val_int, assim como expressar todo e qualquer número em sua forma romana.

Na sequência criamos uma variável de nome numero_romano que será atualizada com o número convertido pelo bloco de código seguinte.

É criada uma variável de controle i, inicialmente de valor 0, pois a validação que devemos fazer no processo de conversão se dá no número de elementos a serem convertidos, uma vez que o usuário irá digitar um determinado número e o processo de conversão nesse caso é feito número por número que compõe o número total digitado.

Sendo assim criamos uma estrutura de repetição while, que enquanto o valor de num for maior que 0, é aplicado um laço for que percorre toda extensão de val_int, validando esse número o dividindo por 0. A cada ciclo de repetição número_romano recebe o valor retornado equivalente em val_rom, compondo o novo valor da variável numero_romano.

Para finalizar apenas são criadas as estruturas para receber do usuário um número via função input( ), exibindo em tela o mesmo convertido.

Supondo que o usuário tenha digitado 1237, o retorno será:

MCCXXXVII

---

84 – Crie um programa que mostra a data atual, formatada para dia-mês-ano, hora:minuto:segundo.

```
import datetime

agora = datetime.datetime.now()

print ("Data e hora atual: ")
print (agora.strftime("%d-%m-%Y, %H:%M:%S"))
```

Uma das bibliotecas mais úteis que temos à disposição para situações que envolvam a manipulação de intervalos de tempo é a biblioteca datetime. Esta por sua vez não vem pré-carregada junto a inicialização padrão do Python, sendo necessário fazer a sua importação por meio do comando import datetime.

Em seguida para que mostremos em tela a hora atual de acordo com o relógio interno do sistema basta chamar a função datetime.now( ) da biblioteca datetime.

Como para nosso exercício vamos aplicar uma formatação à nossa data e hora, atribuímos tal função a nossa variável agora.

Em seguida criamos duas funções print( ), uma exibindo apenas uma mensagem e outra parametrizada com nossa variável agora, aplicando sobre a mesma o método strftime( ) que nos permite formatar manualmente como nossa data e hora serão exibidos.

Nesse caso, para exibir a data no formato como estamos habituados, sendo primeiro o dia, seguido do mês, finalizando com o ano, basta que por justaposição passemos como primeiro parâmetro em strftime( ) %s-%m-%Y, assim como para a hora em formato 24h contando até os segundos, repassamos como segundo parâmetro %H:%M:%S.

O retorno será:

Data e hora atual:

02-02-2021, 18:40:12

---

85 – Crie um programa que exibe a versão atual do Python instalada no sistema:

```
import sys

print("Versão do núcleo Python")
print (sys.version)

print("Versão detalhada:")
print (sys.version_info)
```

Importando a biblioteca sys temos acesso a uma série de funcionalidades que dizem respeito a integração do núcleo Python e sua interação com o sistema operacional, funcionalidades estas que podem ser consultadas na documentação.

Para cumprir o enunciado da questão, inicialmente importamos sys por meio do comando import sys.

Em seguida, apenas por convenção, vamos por meio de algumas funções print( ) exibir em detalhes a versão atual da linguagem Python instalada.

O retorno será:

Versão do núcleo Python
3.6.9 (default, Oct 8 2020, 12:12:24)
[GCC 8.4.0]
Versão detalhada:
Sys.version_info(major=3, minor=6, micro=9, releaselevel='final', serial=0)

---

86 – A partir de uma lista composta apenas de dados numéricos, gere outra lista usando de list comprehension usando como base a lista anterior, compondo a nova com os valores dos elementos originais elevados ao cubo:

```
lista1 = [1, 2, 3, 4, 5, 6]

lista2 = [i ** 3 for i in lista1]

print(lista1)
print(lista2)
```

A chamada compreensão em Python, comumente aplicada a listas e dicionários, nada mais é do que uma forma reduzida e otimizada de se escrever um código sobre esses tipos de dados.

Usando de compreensão podemos usar normalmente de todo e qualquer tipo de dado assim como todo e qualquer tipo de operação lógica ou aritmética.

Dessa forma, inicialmente criamos uma variável de nome lista1 que recebe uma série de números do tipo int como elementos de uma lista.

Na sequência criamos uma nova variável de nome lista2 que, conforme o enunciado, é composta por cada elemento oriundo de lista1 elevado ao cubo.

Sendo assim, o método convencional é por meio de um laço for percorrer cada elemento de lista1, a cada ciclo de repetição elevando o mesmo ao cubo e atribuindo a uma variável temporária i que alimenta cada campo da nova lista, nesse caso lista2.

*Por meio de list comprehension, em lista2 colocamos já de acordo com a sintaxe, colchetes identificando se tratar de uma lista, e dentro da mesma em seu segmento inicial a operação que será aplicada a cada elemento, nesse caso i ** 3 para elevar o mesmo a terceira potência, seguido do laço for, apenas especificando que i é um elemento que deve constar em lista1.*

Dessa forma, lista2 é gerada e tem seu conteúdo atualizado com os valores processados a partir de lista1 pela list comprehension.

O retorno será:
[1, 2, 3, 4, 5, 6]
[1, 8, 27, 64, 125, 216]

---

87 – Dada uma estrutura inicial de um carrinho de compras com 5 itens e seus respectivos valores, assim como uma função que soma os valores dos itens do carrinho, retornando um valor total. Aprimore a função deste código usando de list comprehension:

```
# Estrutura Inicial
carrinho = []
carrinho.append(('Item 1', 30))
carrinho.append(('Item 2', 45))
carrinho.append(('Item 3', 22))
carrinho.append(('Item 4', 93))
carrinho.append(('Item 5', 6))
total = 0
for produto in carrinho:
  total = total + produto[1]
print(f'O valor total é: R$ {total}')
```

```
# Código Otimizado
total2 = sum([y for x, y in carrinho])
print(f'O Valor total é: R$ {total2}')
```

Conforme o enunciado, é fornecida uma estrutura inicial onde simplesmente temos um carrinho, seguido de algumas linhas de código adicionando manualmente itens neste carrinho, finalizando com uma função que soma os valores dos itens, exibindo em tela o valor total.

Normalmente quando falamos de aprimorar/otimizar um código, estamos falando não só de o trazer para a forma sintática mais atual mas também mais otimizada e reduzida.

Em nosso exemplo, claro que por se tratar de um pequeno bloco de código, o impacto em performance na execução de nosso programa será mínimo, mas em aplicações reais blocos de código declarados de forma não otimizada podem sim afetar o desempenho de modo que traga uma experiência ruim ao usuário.

Nesse exercício, como seria possível otimizar um simples bloco de código de uma função? A resposta mais lógica seria

reescrevendo o mesmo usando de comprehension.

Uma vez que o algoritmo em questão, é percorrer os elementos de uma lista, separando-os individualmente e de acordo com sua posição de índice somar os valores dos mesmos retornando esse dado, esse processo pode muito bem ser reescrito por: total2 = sum([y for x, y in carrinho]), onde usamos a função soma do próprio sistema sum( ), por sua vez parametrizado com um laço for que percorre o carrinho, desempacotando apenas os dados de y e somando estes valores.

Por fim, por meio da função print( ) podemos exibir na composição de uma mensagem o último valor da variável total2.

O retorno será:
O valor total é: R$196

---

88 – Escreva uma função que recebe do usuário um número qualquer e retorna para o mesmo tal número elevado ao quadrado. Crie uma documentação para essa função que possa ser acessada pelo usuário diretamente via IDE.

```
def ao_quadrado(num):
    '''Esta função recebe como parâmetro um número,
       retornando ao usuário o resultado do número
       em questão elevado ao quadrado.
       Ex: ao_quadrado(16) retornará 256.
    '''
    return num ** 2

#print(ao_quadrado(18)) retornará 324
print(ao_quadrado.__doc__)
```

Quando estamos falando em documentação, toda e qualquer estrutura em Python possui tanto sua documentação formal acessível pelo site oficial da linguagem, quanto uma documentação reduzida e dinâmica que pode ser consultada diretamente durante a escrita do código.

Uma boa prática de programação é sempre que possível comentar e documentar nossos próprios códigos, principalmente se os mesmos serão distribuídos para terceiros.

Nesse processo, podemos criar um modelo de documentação dinâmica chamada docstring, que por parte de sintaxe simplesmente é escrita como um comentário de múltiplas linhas (comentário identificado por aspas triplas ‘”) logo nas primeiras linhas do corpo de uma função.

Desse modo, ao se instanciar e inicializar uma função, exibindo em tela seu __doc__ via função print( ) será retornada a docstring da mesma.

Em nosso exemplo é criada uma função de nome ao_quadrado( ) que recebe obrigatoriamente como parâmetro um dado/valor para num. Dentro do corpo da função temos uma

docstring, seguido de um retorno padrão definido pela própria expressão matemática que irá pegar o valor de num e elevar ao quadrado.

Nesse caso, parametrizando nossa função print( ) com ao_quadrado.__doc__ o retorno será:

Esta função recebe como parâmetro um número,

Retornando ao usuário o resultado do número

Em questão elevado ao quadrado.

Ex: ao_quadrado(16) retornará 256.

---

89 – Escreva um programa que recebe uma frase qualquer, mapeando a mesma e retornando ao usuário cada palavra com a frequência com que a mesma aparece na frase em questão.

```
frequencia = {}

texto = 'Se alguma coisa pode dar errado, dará errado, e mais, dará errado da pior maneira, no pior momento e de modo que cause o m aior / pior dano possível'

for palavra in texto.split():
    frequencia[palavra] = frequencia.get(palavra, 0) + 1

palavras = frequencia.keys()

for i in palavras:
    print(f'{i} = {frequencia[i]}')
```

Entendendo inicialmente a lógica do exercício, uma vez que temos uma string composta de diversas palavras formando uma frase, será necessário desmembrar e percorrer cada uma dessas palavras para que possamos de fato iterar sobre as mesmas, nesse caso contabilizando sua frequência.

Sendo assim, inicialmente criamos uma variável de nome frequencia que recebe como atributo um dicionário vazio, nela, armazenaremos cada palavra e seu número de incidências.

Em seguida temos uma variável de nome texto que possui como atributo uma das leis de Murphy em forma de string.

Para desmembrar e percorrer cada elemento dessa string, realizamos um laço de repetição for que percorrerá cada elemento desmembrado de texto via texto.split( ), a cada laço de repetição atribuindo seu dado para a variável temporária palavra.

Dentro do corpo desse laço for instanciamos nosso dicionário frequencia, criando a chave [palavra] e atribuindo como valor para mesmo o dado oriundo de frequencia.get( ) parametrizado com

palavra, 0, tendo esse valor acrescido em uma unidade cada vez que uma determinada palavra se repetir ao longo do mapeamento da string.

Por fim é criada uma variável de nome palavras, que recebe apenas os dados de frequencia.keys, ou seja, somente as chaves de nosso dicionário.

Para exibir em tela tais palavras e sua frequência, podemos simplificar o processo com um novo laço de repetição, dessa vez percorrendo o conteúdo da variável palavras, a cada ciclo de repetição atribuindo seu dado/valor para a variável temporária i.

Como ação desse laço de repetição, por meio de nossa função print( ), por sua vez parametrizada com uma f'string, instanciamos em suas máscaras de substituição cada palavra com sua respectiva frequência equivalente.

Nesse caso o retorno será:

Se = 1

alguma = 1

coisa = 1

pode = 1

dar = 1

errado = 3

dará = 2

e = 2

mais, = 1

da = 1

pior = 3

maneira, = 1

no = 1

momento = 1

de = 1

modo = 1

que = 1

cause = 1

o = 1

maior = 1

/ = 1

dano = 1

possível = 1

---

90 – Crie um programa que recebe do usuário uma sequência de números aleatórios separados por vírgula, armazene os números um a um, em formato de texto, como elementos ordenados de uma lista. Mostre em tela a lista com seus respectivos elementos após ordenados.

```
numeros = input('Digite os números, separados por vírgula:')

lista = numeros.split(",")
lista.sort()

print(lista)
```

Aqui temos um problema de simples resolução, bastando desmembrar os números digitados pelo usuário os reordenando de forma crescente.

Inicialmente temos uma variável de nome numeros que por meio da função input( ) recebe do usuário uma sequência de números aleatórios.

Em seguida, é criada uma nova variável de nome lista que simplesmente recebe como atributo o conteúdo de numeros separados por vírgula por meio da função split( ).

Na sequência aplicamos sobre nossa variável lista o método sort( ), que neste caso, sem definirmos nenhum parâmetro específico, simplesmente reordenará os elementos em sua forma crescente.

Por fim, exibimos em tela o conteúdo da variável lista por meio de nossa função print( ).

Nesse caso, supondo que o usuário tenha digitado 9, 2, 5, 6, 3, 7 e 0, o retorno será:

[‘0’, ‘2’, ‘3’, ‘5’, ‘6’, ‘7’, ‘9’]

---

91 – Escreva um programa, da forma mais reduzida o possível, que recebe do usuário uma série de nomes, separando os mesmos e os organizando em ordem alfabética. Em seguida, exiba em tela os nomes já ordenados.

```
nomes = [x for x in input('Digite os nomes, separados por vírgula: ').s
plit(', ')]
nomes.sort()

print(', '.join(nomes))
```

Quando estamos trabalhando em certos contextos onde se faz necessário o uso de códigos reduzidos, temos várias alternativas não somente para otimizar nossos códigos em escrita, mas também no que diz respeito a sua performance.

Para isto, temos desde comprehension até expressões reduzidas que simplificam funções. Em nosso exemplo, uma vez que o usuário tenha dado entrada em vários nomes, precisamos percorrer cada um destes nomes para os separar permitindo assim a reorganização dos mesmos conforme pede o exercício.

Da forma mais reduzida o possível, então criamos uma variável de nome nomes que recebe como atributo uma lista, dentro dessa lista criamos um laço for reduzido que percorrerá cada um dos elementos digitados pelo usuário e retornados por meio da função input( ), separando cada um desses elementos por meio do método split( ) por sua vez parametrizado com ‘, ‘ para que o marcador que identificará a separação de cada um dos elementos seja uma vírgula e espaço, condizendo com a mensagem repassada ao usuário na própria função input( ).

A cada laço, um dos nomes é separado e atribuído a variável temporária x, gerando ao final do processo, cada nome como um dos elementos da lista atribuída a nomes.

Na sequência, para ordenar tais elementos em ordem alfabética basta aplicar o método sort( ), sem parâmetros mesmo, em nossa variável nomes.

Por fim, por meio da função print( ), parametrizada com ‘, ‘;joint( ) por sua vez parametrizada com o conteúdo da variável nomes, novamente estamos concatenando tais nomes para exibi-los em tela, agora reordenados conforme pedido no enunciado da questão.

Supondo que o usuário tenha digitado: Ana, Tania, Paulo, Carlos, Maria, Bibiana, o retorno será:

Ana, Bibiana, Carlos, Maria, Paulo, Tania

---

92 – Escreva um simples programa que recebe do usuário um número qualquer, retornando ao mesmo se este número digitado é um número perfeito.

```
x = int(input('Digite um número: '))

def num_perfeito(x):
    soma = 0
    for i in range(1, x):
        if x % i == 0:
            soma += i
    return soma == x

if num_perfeito(x):
    print(f'{x} é um número perfeito!!!')
else:
    print(f'{x} não é um número perfeito...')
```

Inicialmente temos de relembrar o que é um número perfeito, que na matemática nada mais é do que um número o qual pode ser dividido por todos seus divisores. Ex: O número 6 é um número perfeito, uma vez que o mesmo é divisível por 1, por 3 e por 3, de modo que a soma entre 1, 2 e 3 é igual a 6.

Sendo assim, indo para o código, de início criamos uma variável de nome x que recebe, já validando como número inteiro via int( ) um número digitado pelo usuário por meio da função input( ).

Em seguida é criada a função num_perfeito( ) que recebe o conteúdo da variável x como próprio parâmetro.

No corpo dessa função é criada uma variável de nome sima, inicialmente definida com valor 0.

Na sequência, é criado um laço for que percorre um intervalo entre 1 até o valor de x, haja visto que uma das validações que teremos de realizar para reconhecer um número como perfeito é o mesmo ser divisível por seus antecessores.

Logo, nesse laço de repetição criamos uma estrutura condicional que verifica se a divisão inteira entre o valor de x e cada elemento percorrido pelo laço for é igual a 0. Caso essa condição seja verdadeira, o calor de soma é atualizado somando seu valor atual com o valor de i. Caso contrário, soma apenas tem seu valor atualizado com o valor de x.

Dentro dessa lógica, com um laço simples e uma condicional simples conseguimos realizar todas validações necessárias para verificar um número como perfeito ou não.

Por fim, apenas criamos uma estrutura condicional onde se o retorno de num_perfeito( ) por sua vez parametrizado com o valor de x for True, é exibido em tela uma mensagem dizendo que tal valor é perfeito. Do mesmo modo, caso a condição anterior não seja verdadeira, é exibida em tela uma mensagem dizendo que o valor de x não é um número perfeito.

Supondo que o usuário tenha digitado 6, o retorno será:

6 é um número perfeito!!!

Supondo que o usuário tenha digitado 8, o retorno será:

8 não é um número perfeito...

---

93 – Escreva uma função que recebe uma lista de elementos totalmente aleatórios e os ordena de forma crescente de acordo com seu valor numérico:

```
def ordena_lista(lista):
    for i in range(1, len(lista)):
        valor = lista[i]
        indice = i -1
        while indice >= 0:
            if valor < lista[indice]:
                lista[indice + 1], lista[indice] = lista[indice], lista[indice + 1]
                indice -= 1
            else:
                break
    return lista

lista1 = ordena_lista([9, 0, 3, 5, 1, 6, 7, 2, 8, 4])

print(lista1)
```

Para certos contextos, trabalhar com listas de elementos ordenados se faz necessário, logo, é necessário escrevermos um bloco de código dedicado a este propósito, seja atualizando uma lista já existente, seja gerando uma nova lista ordenada a partir de uma lista de elementos desordenados.

Partindo para o código, já de início criamos uma função de nome ordena_lista( ) que recebe como parâmetro o conteúdo de uma lista.

No corpo dessa função inicialmente é necessário criar uma estrutura de código que irá percorrer os elementos da lista, e isso é feito por meio de um laço for, que percorrerá um intervalo entre 1 até o tamanho total da lista por meio de len( ) parametrizado com lista.

A cada ciclo de repetição de nosso laço, um elemento da lista tem seu valor lido e atribuído a variável temporária i.

Dentro do laço, é criada uma variável de nome valor que recebe lista na posição i. De forma parecida, uma variável de nome índice receberá i -1, ou seja, para cada elemento lido em i será atribuído um valor, lembrando que índices em Python por convenção são iniciados em 0.

Ainda dentro do laço for é criada uma estrutura condicional onde enquanto o valor de índice for igual ou maior que 0 (apenas validando que um índice não pode ter valores abaixo de 0 ou "negativos").

Nessa condicional é criado uma outra estrutura condicional que verifica se o valor atribuído a variável valor for menor que lista na posição índice, as variáveis responsáveis por guardar os dados da lista de acordo com seu índice são atualizadas; lista em sua posição índice + 1 recebe o último atribuído para lista em sua posição índice, da mesma forma lista em sua posição índice tem seu valor atualizado para o último valor atribuído a lista na posição índice + 1. Por fim índice tem seu valor decrementado em uma unidade e desse modo temos criado o mecanismo de ordenação dos elementos da lista, rearranjando os mesmos de forma crescente.

Quando a estrutura condicional definida anteriormente já não se fizer mais válida então o laço de repetição é interrompido, retornando o último valor atribuído a lista, que nesse caso, será a mesma com todos seus elementos agora ordenados.

Já fora da função é criada uma variável de nome lista1 que diretamente chama a função ordena_lista( ) parametrizando a mesma com a lista[9, 0, 3, 5, 1, 6, 7, 2, 8, 4].

Por meio de nossa velha conhecida função print( ) parametrizada com lista1 o que ocorre é um aninhamento de funções uma vez que a própria variável lista1 possui como atributo a função ordena_lista( ).

Nesse caso, não havendo nenhum erro de sintaxe o retorno será:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

---

94 – Crie uma estrutura toda orientada a objetos que recebe do usuário uma string qualquer, retornando a mesma com todas suas letras convertidas para letra maiúscula. Os métodos de classe para cada funcionalidade devem ser independentes entre si, porém trabalhar no escopo geral da classe.

```
class Conversor(object):
    def __init__(self):
        self.s = ""

    def recebe_string(self):
        self.s = input('Digite uma palavra/frase: ')

    def exibe_string(self):
        print(self.s.upper())

frase1 = Conversor()
frase1.recebe_string()
frase1.exibe_string()
```

Seguindo as orientações do enunciado da questão, inicialmente criamos uma classe de nome Conversor que receberá um object.

Dentro do corpo dessa classe inicialmente é criado um método construtor/inicializador __init__( ), dentro do mesmo é criado um atributo de classe de nome s que simplesmente recebe uma string vazia.

Seguindo a indentação, é criado um método de classe de nome recebe_string( ), que pode receber um objeto qualquer, retornando o mesmo para o escopo geral da classe.

No corpo desse método de classe é instanciado o atributo de classe s, atribuindo para o mesmo via função input( ) alguma string digitada pelo usuário.

De forma parecida é criado um novo método de classe de nome exibe_string( ), que por sua vez, obedecendo o enunciado da

questão, exibe em tela via print( ) o valor de s, tendo todos seus caracteres convertidos para letra maiúscula por meio do método upper( ).

De volta ao escopo global do código, é criada uma variável de nome frase1 que instancia a classe Conversor( ). A partir desse ponto, aplicando para frase1 o método recebe_string( ) será pedido que o usuário digite uma palavra ou frase, assim como aplicando o método exibe_string( ) a palavra/frase digitada pelo usuário será exibida em letras maiúsculas.

Nesse caso, digitando Python Fernando Feltrin o retorno será:

PYTHON FERNANDO FELTRIN

---

95 – Escreva de forma reduzida um programa que recebe do usuário um nome e duas notas, salvando tais dados como um elemento de uma lista. Exiba em tela o conteúdo dessa lista.

```
from operator import itemgetter

nomes = []

while True:
    dados = input('Digite seu nome, seguido de duas notas de 0 a 10:
')
    if not dados:
        break
    nomes.append(tuple(dados.split(",")))

print(sorted(nomes, key = itemgetter(0,1,2)))
```

Como bem sabemos, em Python sempre existem muitas formas diferentes de escrever um código para um determinado propósito. De acordo com o enunciado da questão, para salvar como um elemento de lista vários dados de diferentes tipos podemos usar de diversos contêineres de dados, um dos mais eficientes para contextos onde temos poucos dados a serem armazenados são as tuplas. Trabalhando com tuplas podemos usar de um recurso interessante chamado itemgetter da biblioteca operator para instanciar e se trabalhar apenas com um dos elementos de uma lista sem alterar suas propriedades e comportamentos.

Nessa linha de raciocínio, inicialmente importamos da biblioteca operator o módulo itemgetter para fazer uso de seus recursos.

Em seguida é criada uma lista de nome nomes, que inicialmente recebe como atributo uma lista vazia.

Na sequência, para criar o mecanismo que recebe do usuário diferentes nomes com suas respectivas notas, podemos fazer o uso

de um laço de repetição, onde enquanto válido, uma variável de nome dados recebe via input( ) do usuário os respectivos dados de acordo com o pedido na questão.

Indentado dentro da estrutura de repetição é criada uma estrutura condicional if que verifica se não houverem novos dados atribuídos a dados (supondo que o usuário não tenha digitado mais nada e tenha pressionado ENTER pela última vez), se encerra o ciclo de repetição via break.

Cada um dos dados digitados pelo usuário, em cada ciclo de repetição, é adicionado via append( ) para nomes, convertendo tais dados como elementos de uma tupla, separando os mesmos via split( ) usando como marcador de referência onde houver vírgula.

De volta ao corpo geral do código, podemos diretamente por meio da função print( ), parametrizada com sorted( ), por sua vez parametrizada em justaposição com nomes, que é a origem dos dados, sendo para o parâmetro key atribuída a função itemgetter( ), com os marcadores de posição de índice 0, 1 e 2.

Dessa forma, de uma maneira simples, porém elaborada, estaremos interagindo com cada elemento da lista, desempacotando de tal elemento os itens das respectivas posições de índice definidas anteriormente.

Supondo que o usuário tenha digitado Fernando, 8, 9 e Maria, 9, 8, o retorno será:

[(‘Fernando’, ‘8’, ‘9’), (‘Maria’, ‘9’, ‘8’)]

---

96 – Crie um programa que gera o valor de salário de funcionários considerando apenas horas trabalhadas e horas extras, sendo o valor fixo da hora trabalhada R$29,11 e da hora extra R$5,00. Crie uma regra onde o funcionário só tem direito a receber horas extras a partir de 40 horas trabalhadas da forma convencional.

```
valor_hora = 29.11
valor_hora_extra = 5

horas = int(input('Digite o número de horas trabalhadas: ')) * valor_h
ora
adicional = input('Digite o número de horas extras: ') * valor_hora_ex
tra

if horas > 40:
    valor_final = horas + adicional
else:
    valor_final = horas

print(f'Salário base: R${horas}')
print(f'Adicional de horas extras: R${adicional}')
print(f'Remuneração Total: R${valor_final}')
```

De acordo com o enunciado da questão, basicamente teremos de gerar um valor em reais, simulando o salário de um determinado funcionário de acordo com algumas regras.

Em nosso código, inicialmente vamos definir as regras, sendo uma variável de nome valor_hora que recebe seu respectivo valor de 29.11, da mesma forma a variável valor_hora_extra recebe seu respectivo valor 5.

Uma vez definidas as variáveis que regrarão os cálculos, podemos trabalhar a parte lógica da questão.

Para isso criamos uma variável de nome horas que recebe do usuário via input( ), já validando como número inteiro via int( ), um

determinado número de horas trabalhadas. Da mesma forma é criada uma variável de nome adicional que recebe o número referência para as horas extras.

Na sequência é criada uma estrutura condicional que verifica se o valor de horas é superior a 40, de acordo com a regra, caso tal condição seja válida, uma nova variável de nome valor_final recebe como atributo a soma dos valores de horas e de adicional. Caso a condição imposta não seja válida, valor final recebe apenas o valor referente as horas.

Por fim, podemos via print( ) criar alguns retornos elaborados, sendo no primeiro deles, fazendo uso de f'strings, exibindo em tela o valor de horas, o segundo deles o valor de adicional e o terceiro deles o valor_final contextualizados com suas respectivas mensagens.

Supondo que o usuário tenha digitado 68 e 166, respectivamente, o retorno será:

Salário base: R$1979.48

Adicional de horas extras: R$830.0

Remuneração Total: R$2809.48

---

97 – Reescreva o código anterior adicionando um mecanismo simples de validação que verifica se os dados inseridos pelo usuário são de tipos numéricos, caso não sejam, que o processo seja encerrado.

```
valor_hora = 29.11
valor_hora_extra = 5

try:
    horas = int(input('Digite o número de horas trabalhadas: ')) * valor
_hora
    adicional = float(input('Digite o número de horas extras: ')) * valor_
hora_extra

    if horas > 40:
        valor_final = horas + adicional
    else:
        valor_final = horas

except:
    print('Digite apenas números...')
    quit()

print(f'Salário base: R${horas}')
print(f'Adicional de horas extras: R${adicional}')
print(f'Remuneração Total: R${valor_final}')
```

Uma boa prática de programação é, ao longo de nossos códigos, criar certos mecanismos de validação para evitar possíveis exceções cometidas pelo usuário.

Nessa linha de raciocínio, reaproveitando o código anterior podemos simplesmente via try e except rearranjar alguns blocos de código de modo que os mesmos somente serão executados caso nenhuma exceção seja cometida.

Sendo assim, logo em nossa quarta linha de código é inserido um try, onde dentro do corpo do mesmo constam todas as

estruturas tanto para receber os dados do usuário quanto para calcular os devidos valores do salário baseado em tais dados.

Dessa forma, logo nas primeiras linhas do bloco indentado para try, caso o tipo de dado atribuído para horas ou para adicional sejam diferentes de int e float, respectivamente, essa estrutura condicional se torna inválida, não executando nenhuma linha subsequente desse bloco de código, pulando diretamente para o except, que por sua vez exibe uma mensagem de erro via print( ) e encerra totalmente o processo via quit( ).

Caso o usuário digite os valores corretos como esperado, todos os cálculos são realizados, exibindo em tela por meio de nossas funções print( ) declaradas ao final do códigos os respectivos valores de salário.

Supondo que o usuário tenha digitado 68 e 166, o retorno será:

Salário base: R$1979.48

Adicional de horas extras: R$830.0

Remuneração Total: R$2809.48

Simulando o erro, supondo que o usuário tenha digitado 68 e h, o retorno será:

Digite apenas números...

---

98 – Crie um programa que recebe uma nota entre 0 e 1.0, classificando de acordo com a nota se um aluno fictício está aprovado ou em recuperação de acordo com sua nota. A média para aprovação deve ser 0.6 ou mais, e o programa deve realizar as devidas validações para caso o usuário digite a nota em um formato inválido.

```
def calcula_nota(nota):
    if nota < 0 or nota > 1.0:
        print('Pontuação inválida!!!')
        print('A nota deve ser um valor entre 0 e 1.0')
    elif nota == 1.0:
        print('Parabéns, você gabaritou a prova!!!')
    elif nota >= 0.6:
        print('Parabéns, você foi aprovado(a)!!!')
    elif nota < 0.6:
        print('Nota abaixo da média, você está em recuperação!!!')
    else:
        print('Não foi possível computar sua nota!!!')

try:
    nota = float(input('Digite uma nota: '))
except:
    print('Valor inválido!!!')
    print('Use somente números com casas decimais entre 0 e 1.0')
    quit()

print(calcula_nota(nota))
```

Para um exemplo como este basta que usemos da forma correta estruturas condicionais para realizar tanto a classificação por nota quanto as validações de possíveis erros a serem cometidos pelo usuário. Para isso, podemos criar um código bastante enxuto fazendo o uso de if, elif, else, try e except.

Inicialmente criamos uma função responsável por classificar a nota, função essa de nome calcula_nota( ) que recebe como

parâmetro uma nota.

Já dentro do corpo da função, logo na primeira linha já criamos nossa primeira estrutura condicional que verifica se o valor repassado como note é menor que 0 ou maior que 1.0, sendo válida essa condição, é exibido ao usuário uma mensagem de erro via print( ) orientando o mesmo a usar o programa da maneira correta.

Em seguida criamos uma estrutura elif, que verifica se o valor de nome é igual a 1.0, exibindo em tela via print( ) que o aluno gabaritou a prova.

Mais um elif é criado, dessa vez verificando se o valor de nota é igual ou maior a 0.6, caso essa afirmação seja válida, é exibido em tela via print( ) uma mensagem parabenizando o usuário por sua aprovação.

Outro elif é criado, agora verificando se o valor de nota é menor que 0.6, nesse caso exibindo em tela para o usuário uma mensagem dizendo ao mesmo que com base em sua nota está em recuperação.

Por fim é criado um else, apenas exibindo em tela uma mensagem de erro a ser exibida quando nenhuma das possibilidades anteriores forem válidas.

Dessa forma, de acordo com a nota digitada apenas um dos ifs/elifs será validado exibindo em tela sua respectiva mensagem.

De volta ao corpo geral do código, podemos via try criar uma outra estrutura de validação incorporada diretamente sobre a estrutura que receberá via input( ) alguma nota digitada pelo usuário, dessa vez simultaneamente verificando se o valor atribuído a variável nota é do tipo float.

Caso essa condição imposta não seja atendida, é exibido em tela via print( ) algumas mensagens de erro, encerrando o processo via quit( ).

Terminadas as estruturas lógicas de nosso programa, podemos diretamente via função print( ) chamar nossa função

calcula_nota( ) por sua vez parametrizada com o valor atribuído a variável nota.

Supondo que o usuário tenha digitado 7, o retorno será:

Pontuação inválida!!!

A nota deve ser um valor entre 0 e 1.0

Supondo que o usuário tenha digitado 0.7, o retorno será:

Parabéns, você foi aprovado(a)!!!

Supondo que o usuário tenha digitado k, o retorno será:

Valor inválido!!!

Use somente números com casas decimais entre 0 e 1.0

---

99 – Crie uma estrutura molde (orientada a objetos) para cadastro de veículos, tendo como características que os descrevem sua marca, modelo, ano, cor e valor. Cadastre ao menos três veículos, revelando seu número identificador de objeto alocado em memória, assim como o retorno esperado pelo usuário quando o mesmo consultar tal veículo.

```
class Carro:
    def __init__(self, marca = None, modelo = None, ano = None, cor
= None, valor = None):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano
        self.cor = cor
        self.valor = valor

    def descricao(self):
        descricao_carro = f'{self.marca}, {self.modelo}, {self.ano}, {self.c
or}, {self.valor}'
        return descricao_carro

carro1 = Carro('Audi', 'A3', 2006, 'Preto', 'R$ 19.900')
carro2 = Carro('Renault', 'Megane', 2010, 'Preto', 'R$ 22.500')
carro3 = Carro('Chevrolet', 'Cruise', 2018, 'Vermelho', 'R$ 49.000')

print(carro1)
print(carro2)
print(carro3)

print(carro1.descricao())
print(carro2.descricao())
print(carro3.descricao())
```

De acordo com o enunciado, para este exercício devemos desenvolver um simples programa para cadastro de veículos, tendo sua estrutura base orientada a objetos para que possamos a

reutilizar como molde para cadastro de novos veículos. Basicamente, aqui estaremos trabalhando com manipulação simples de objetos atributos de classe.

Sendo assim, já de início podemos criar tal estrutura, criando uma classe de nome Carro, onde em seu método construtor / inicializador __init__( ) podemos criar objetos referentes as características do veículo como marca, modelo, ano, cor e valor através de seu construtor self.nome_do_objeto, assim como em seu próprio campo de argumentos atribuindo um valor padrão None para validar casos onde não teremos todas as características do veículo no momento da geração de seu cadastro.

Ainda dentro da classe criamos um método de classe de nome descricao( ), que retorna uma descrição do veículo com base em suas características.

No corpo geral do código são criadas três variáveis, carro1, carro2 e carro3, que instanciam a classe Carro, repassando para a mesma por justaposição algumas características de veículos.

De acordo com o enunciado da questão, para exibirmos o identificador de cada objeto, basta parametrizar o mesmo em nossa função print( ), pois se tratando de uma estrutura orientada a objetos, o retorno padrão ao se instanciar um objeto/variável será seu número identificador e não seu conteúdo.

Para revelar o conteúdo de cada objeto, ao usar o mesmo como parâmetro de nossa função print( ) devemos instanciar também o método de classe responsável por retornar tais dados, nesse caso o método de classe descricao( ).

Não havendo nenhum erro de sintaxe, o retorno será:

<__main__.Carro object at 0x0000018EFD30D190>

<__main__.Carro object at 0x0000018EFD30DB80>

<__main__.Carro object at 0x0000018EFD4270A0>

Audi, A3, 2006, Preto, R$ 19.900

Renault, Megane, 2010, Preto, R$ 22.500

Chevrolet, Cruise, 2018, Vermelho, R$ 49.000

---

100 – Crie um programa que recebe do usuário três números diferentes, retornando para o mesmo qual destes números é o de maior valor. Usar de funções aninhadas para realizar as devidas comparações entre os valores dos números:

```
x = int(input('Digite o primeiro número: '))
y = int(input('Digite o segundo número: '))
z = int(input('Digite o terceiro número: '))

def maior_de_dois(x, y):
    if x > y:
        return x
    return y

def maior_de_tres(x, y, z):
    return maior_de_dois(x, maior_de_dois(y, z))

print(f'O maior número entre {x}, {y} e {z} é: {maior_de_tres(x, y, z)}')
```

Quando estamos trabalhando com expressões lógicas é comum sempre haver diversas formas de solucionar o mesmo problema, cabendo ao desenvolvedor adotar o método o qual julgue o mais simples e funcional.

Nessa linha de raciocínio, uma vez que temos um problema lógico de verificar valores numéricos e extrair informações a partir dos mesmos, uma boa prática de programação (por parte de eficiência) é usar de funções aninhadas, funções que interagem entre si, umas chamando as outras.

Para nosso exemplo, inicialmente declaramos três variáveis de nomes x, y e z que por meio da função input( ) pedem que o usuário digite um número que será atribuído para cada uma destas variáveis.

Na sequência, criamos uma função de nome maior_de_dois( ) que recebe dois números, repassados em forma de parâmetro, oriundos das variáveis x e y.

Dentro do corpo dessa função simplesmente criamos uma estrutura condicional que se o valor de x for maior que o valor de y, é retornado o valor de x, caso contrário, é retornado o valor de y.

Do mesmo modo, criamos uma função de nome maior_de_tres( ), que receberá três parâmetros, x, y e z.

No corpo dessa função diretamente criamos um retorno que instancia e inicializa a função maior_de_dois( ), por sua vez parametrizada em justaposição com x para seu parâmetro x, e com a função maior_de_dois( ) parametrizada com y e z no lugar do parâmetro y.

Dessa forma, de acordo com a estrutura condicional interna a maior_de_dois( ), temos as devidas interações e comparações entre os valores das três variáveis em questão.

Finalizando, diretamente em nossa função print( ), usando de uma f'string, podemos em suas máscaras de substituição instanciar as variáveis x, y e z, assim como chamar a função maior_de_tres( ), por sua vez parametrizada com os dados de x, y e z. Lembrando que caso fosse necessário guardar essa informação, deveríamos atribuir a função maior_de_tres( ) para alguma variável.

Supondo que o usuário tenha digitado 1, 8 e 5, respectivamente (embora a ordem não importe nesse caso), o retorno será:

O maior número entre 1, 8 e 5 é 8

---

101 – Crie um programa que atua como mecanismo controlador de um sistema direcional, registrando a direção e o número de passos executado. Ao final do processo, exiba em tela o número de referência para um plano cartesiano. Ex: Se o usuário digitar CIMA 5, BAIXO 3, ESQUERDA 3 e DIREITA 2, o resultado será a coordenada 2.

```
import math

posicao = [0,0]

while True:
    coordenada = input('Digite a coordenada, seguido do número de p
assos: ')
    if not coordenada:
        break

    movimento = coordenada.split(" ")
    direcao = movimento[0]
    passos = int(movimento[1])

    if direcao == 'CIMA':
        posicao[0] += passos
    elif direcao == 'BAIXO':
        posicao[0] -= passos
    elif direcao == 'ESQUERDA':
        posicao[1] -= passos
    elif direcao == 'DIREITA':
        posicao[1] += passos
    else:
        pass

print(int(round(math.sqrt(posicao[1] ** 2 + posicao[0] ** 2))))
print(posicao)
```

Para esta questão, temos um problema que a primeira vista pode parecer algo complexo, porém nada mais é do que um sistema que receberá do usuário coordenadas digitadas, mapeando as mesmas e as transformando em um referencial de uma coordenada, de modo que podemos abstrair tal problema em blocos de estruturas condicionais simples.

Para isso, inicialmente importamos a biblioteca math para fazer uso de suas funções.

Na sequência criamos uma variável de nome posicao, que por sua vez recebe em formato de lista dois valores que podemos imaginar como coordenadas x e y, uma vez que o enunciado da questão nos pede coordenadas de um plano cartesiano.

Em seguida criamos uma estrutura while True, dentro do bloco da mesma é criada uma variável de nome coordenada, que via input( ) recebe do usuário uma coordenada e um valor para o número de passos.

Também criamos uma estrutura condicional onde quando não houverem mais dados atribuídos a coordenada, se encerra esse processo. Dessa forma, podemos receber um número ilimitado de coordenadas do usuário, encerrando o processo quando o mesmo não digitar novas coordenadas.

Dando sequência, criamos uma variável de nome movimento, que recebe os dados de coordenada separados por espaço via split( ).

É criada uma variável de nome direção que recebe o valor do elemento de posição 0 da variável movimento. Do mesmo modo é criada uma variável de nome passos, que recebe o dado/valor de posição 1 da variável movimento.

Na sequência é necessário criar a estrutura lógica que armazenará os dados de direção e passos que o usuário forneceu.

Isto é feito por uma série de if, elif e else, sendo o primeiro if verificando se o valor atribuído a variável direcao é igual a 'CIMA',

nesse caso atualizando o valor da variável posicao em sua posição 0 somado com o valor de passos.

De forma parecida é criado um elif que verifica se o valor de direcao é igual a 'BAIXO', caso seja, o valor da variável posicao na posição 0 de índice tem seu valor decrementado com o valor de passos.

Mais um elif é criado, agora verificando se o valor de direcao é igual a 'ESQUERDA', caso sim, o valor da variável posicao em sua posição de índice 1 é decrementado com o valor de passos.

Por fim, um último elif que verifica se o valor de direcao é igual a 'DIREITA', caso for, a variável posicao tem seu valor na posição 1 de índice incrementada com o valor de passos.

Finalizando esta parte, é definido um else pass para que caso o usuário digite uma direção ou número de passos inválido nada aconteça.

Uma vez terminada toda a parte lógica do código podemos exibir em tela a coordenada final por meio de nossa função print( ), parametrizada com a raiz da soma dos valores da variável posicao em suas posições de índice 0 e 1 via math.sqrt( ), tendo seu valor final arredondado via round( ).

Da mesma forma, podemos exibir em tela os últimos valores atribuídos para a variável posicao usando a mesma como parâmetro de nossa função print( ).

Simulando exatamente os mesmos passos descritos no enunciado da questão, CIMA 5, BAIXO 3, ESQUERDA 3 e DIREITA 2, o resultado deverá ser a coordenada 2. Nesse caso, o retorno será:

2

[2, -1]

---

102 – Crie uma estrutura orientada a objetos híbrida, para cadastro de usuários, que permita ser instanciada por uma variável de modo que a mesma pode ou não repassar argumentos para seus objetos de classe na hora em que instancia tal classe. Para isso, a classe deve conter atributos/objetos de classe padrão e instanciáveis de fora da classe.

```
class Usuario:
    nome = 'usuário'
    senha = 'padrao'

    def __init__(self, nome = None, senha = None):
        self.nome = nome
        self.senha = senha

usuario1 = Usuario('admin', '00237')
print(f'Usuário registrado: {Usuario.nome}')
```

Aqui temos um exemplo clássico de estrutura orientada a objetos que servirá como molde para criação de novos objetos a partir da mesma, tendo em vista sua estrutura de modo a oferecer compatibilidade para diferentes formas de uso por parte do usuário.

Partindo para o código, inicialmente criamos uma classe de nome Usuario, dentro do corpo da mesma, logo nas primeiras linhas e antes mesmo do método construtor são criados os objetos nome e senha com atributos padrão em formato de string. Desse modo, temos objetos que podem tanto serem instanciáveis por métodos de classe, quanto por métodos estáticos, independentemente do escopo.

Na sequência é criado um método construtor / inicializador __init__( ) que receberá um nome e uma senha, já validando as mesmas em seus campos com valor padrão None, para caso o usuário use dos objetos nome e senha do escopo geral da classe,

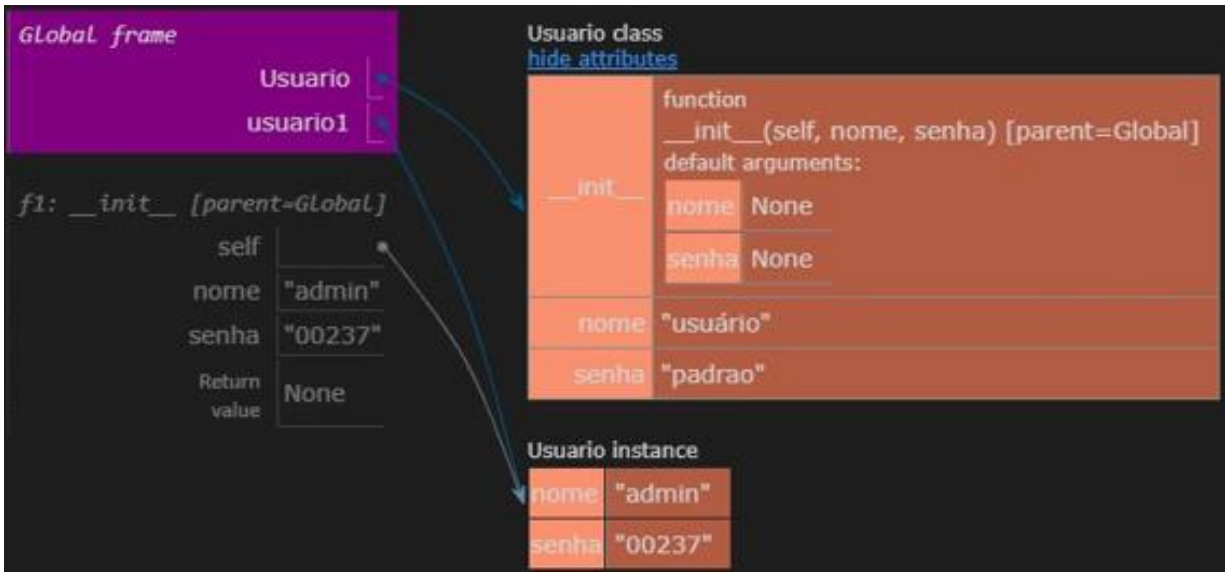
os objetos nome e senha do método construtor não sejam inicializados.

De volta ao escopo global do código, é criada uma variável de nome usuario1, que instancia a classe Usuário, repassando como argumentos para a mesma em justaposição 'admin' e '00237' para nome e senha. Dessa forma, a variável usuario1 está criando seus objetos atributos de classe fazendo o uso dos objetos do método construtor da classe.

Por meio da função print( ), fazendo o uso de uma f'string, podemos em sua máscara de substituição instanciar Usuario.nome, internamente o interpretador fará referência ao atributo de classe nome situado no método construtor.

Nesse caso o retorno será:

Usuário registrado: admin



---//---

```
class Usuario:
    nome = 'usuário'
    senha = 'padrao'

    def __init__(self, nome = None, senha = None):
        self.nome = nome
```

```
        self.senha = senha

usuario2 = Usuario()
usuario2.nome = 'tecnico'
usuario2.senha = '12345'
print(f'Usuário registrado: {usuario2.nome} {usuario2.senha}')
```

Simulando a outra possível modalidade de interação, no corpo geral do código é criada uma variável de nome usuario2, que por sua vez apenas instancia a classe Usuario.

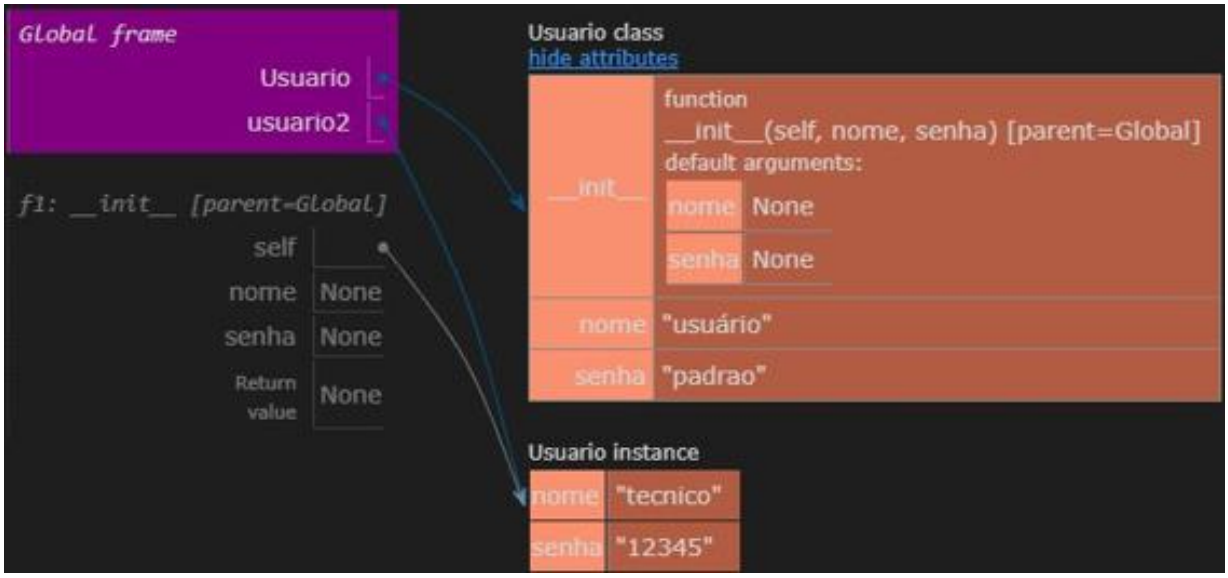
Em seguida, é definido que usuario2.nome tem seu dado/valor de atributo a string 'tecnico'. Da mesma forma é atribuído para usuario2.senha a string '12345'.

Nesse caso, toda a interação está ocorrendo instanciando as variáveis nome e senha do escopo geral da classe, e não do escopo do método inicializador.

Por meio da função print( ), exatamente da mesma forma como no exemplo anterior, é possível em uma máscara de substituição de uma f1string incorporar os dados de usuario2.nome e usuario2.senha, sendo que agora o interpretador irá inicializar as variáveis nome e senha apenas do escopo global da classe.

Nesse caso, o retorno será:

Usuário registrado: técnico 12345


Global frame
Usuario
usuario2
f1: __init__ [parent=Global]
self
nome
None
senha
None
Return value
None
Usuario class
hide attributes
__init__
function
__init__(self, nome, senha) [parent=Global]
default arguments:
nome
None
senha
None
nome
"usuário"
senha
"padrao"
Usuario instance
nome
"tecnico"
senha
"12345"

---

103 – Gere uma lista com 50 elementos ordenados de forma crescente. Inicialmente usando do método convencional (e do construtor de listas padrão), posteriormente reescrevendo o código usando de list comprehension.

```
lista1 = list()

for num in range(51):
    lista1.append(num)

print(lista1)
```

Resolvendo o problema proposto pela questão em seu método convencional, inicialmente criamos uma variável de nome lista1 que simplesmente instancia o método construtor de listas list( ), sem parâmetros mesmo.

Na sequência, por meio de um laço for e do método range( ) podemos gerar os elementos ordenados para nossa lista. Sendo que a cada laço de repetição um elemento é adicionado a nossa lista1 via append( ), delimitando o número de elementos no método range( ) em 51 uma vez que o último elemento dessa função é apenas o gatilho para o encerramento do intervalo numérico, não tendo seu valor contabilizado.

Por fim, via função print( ), por sua vez parametrizada com o conteúdo da variável lista1, exibiremos em tela os elementos da mesma.

Nesse caso o retorno será:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]

---//---

```
lista2 = [num for num in range(51)]

print(lista2)
```

Reescrevendo o código de forma reduzida, usando de list comprehension, para nossa variável lista2 atribuímos uma lista onde é usado um laço for percorrendo cada elemento do intervalo numérico definido como parâmetro para range( ), retornando cada um desses elementos para a variável temporária num.

Lembrando que a ordem dos elementos em comprehension é lida sendo o primeiro num sendo o elemento final da lista, o segundo num a variável temporária do laço for e, nesse caso, range( ) a função a ser usada (podendo também nesse campo ser usado de alguma expressão lógica ou matemática de acordo com o propósito).

Por meio da função print( ), novamente podemos exibir o conteúdo de nossa lista, nesse caso, lista2.

Não havendo nenhum erro de sintaxe, o retorno será:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]

---

104 – Reescreva a estrutura de código abaixo, de um simples programa que gera uma lista com as possíveis combinações entre duas outras listas [1,2,3] e [4,5,6], reduzindo e otimizando a mesma via list comprehension:

```
lista1 = []

for x in [1, 2, 3]:
    for y in [4, 5, 6]:
        if x != y:
            lista1.append((x, y))

print(lista1)
```

Para fins de conferência, o retorno deverá ser:

[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]

```
lista2 = [(x, y) for x in [1,2,3] for y in [4,5,6] if x != y]

print(lista2)
```

Declarada uma variável de nome lista2, podemos partir direto para o processo de otimização.

Nesse caso, criando um primeiro laço for que percorrerá cada elemento de [1,2,3], retornamos a cada ciclo de repetição um dado/valor para x. Exatamente da mesma forma um segundo laço for percorrerá cada um dos elementos de [4,5,6], retornando seus valores para variável temporária y.

Como parte final da estrutura de comprehension, criamos uma estrutura condicional que simplesmente verifica se o último valor atribuído a x é diferente de y, sendo essa condição verdadeira, retornando seus respectivos valores para (x, y) escritos como primeiro elemento da sentença.

Dessa forma, embora um pouco confusa para iniciantes, reduzimos e otimizamos o mesmo processo escrito de forma

convencional para o enunciado da questão.

Por meio de nossa função print( ), exibindo o conteúdo da variável lista2, podemos ver que de fato o processo foi realizado com sucesso, haja visto que o resultado retornado é o mesmo do resultado apresentado para conferência na própria questão.

[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]

---

105 – Escreva um programa que cria uma array de 5 elementos do tipo int. Exiba em tela todos os elementos da array, em seguida exiba individualmente apenas os elementos pares da mesma de acordo com sua posição de índice.

```
from array import *

numeros = array('i', [5, 10, 15, 20, 25])

for i in numeros:
    print(i)
    if i % 2 == 0:
        print(f'{i} é um número par')
```

Para criarmos uma array (tipo de dado array) conforme o exercício pede, importamos todo o conteúdo do módulo array.

Em seguida criamos uma variável de nome numeros que usando do método construtor de arrays array( ) cria atribuído para si uma array de tipo de dados internos definidos pelo primeiro parâmetro em justaposição, seguido da lista de elementos em si, nesse caso, 5 valores inteiros múltiplos de 5 apenas para fins de exemplo.

Para exibirmos os elementos da array, uma das formas mais básicas é usar de um laço for que percorrerá todos elementos da array, a cada ciclo atribuindo seu valor à variável temporária i, exibindo seu conteúdo em tela via função print( ).

Por fim, para dar destaque aos elementos pares da array, podemos diretamente dentro do laço for criar uma estrutura condicional que realiza a verificação se um determinado número é par simplesmente validando se o módulo (resto da divisão) do mesmo por 2 é igual a zero. Caso a condição imposta seja verdadeira, é exibida uma mensagem personalizada em tela via função print( ).

Nesse caso o retorno será:

```
5
10
10 é um número par
15
```

20
20 é um número par
25

---

106 – Crie uma array composta de 6 elementos aleatórios, porém com valores sequenciais crescentes. Uma vez criada a array, exiba em tela seu conteúdo em seu formato original, e em seu formato invertido.

```
from array import *

numeros = array('i', [1, 2, 4, 8, 16, 32])

for i in numeros:
    print(i)

numeros.reverse()

for i in numeros:
    print(i)
```

Seguindo com a mesma lógica do exercício anterior, podemos criar uma array usando do método array( ), por sua vez parametrizado com o tipo de dados dos elementos e da lista de elementos em si.

Na sequência é criado um laço for que percorre todos os elementos da array atribuída a variável numeros, exibindo um a um a cada ciclo de repetição.

Para invertermos a posição dos elementos da array, podemos simplesmente instanciar a variável numeros aplicando sobre a mesma o método reverse( ), sem parâmetros mesmo.

Novamente fazendo uso de um laço de repetição, podemos percorrer e exibir em tela via print( ) cada um dos elementos que compõem a array.

O retorno será:

1
2
4
8
16
32
32

16
8
4
2
1

107 – Dada a seguinte lista simples de elementos [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21], transforme a mesma em um objeto do tipo array, em seguida exiba em tela o tamanho dessa array (número de elementos), separadamente, exiba o tamanho da array em bytes e seu marcador de posição alocada em memória.

```
from array import *

lista1 = array('i', [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21])

print(f'lista1 é composta de {len(lista1)} elementos')

print(f'lista1 possui um tamanho de {lista1.itemsize} bytes')

print(f'lista1 está alocada na memória no endereço: {lista1.buffer_info()[0]}')
```

Uma vez importado todo o conteúdo do módulo array, podemos fazer uso de suas ferramentas conforme necessário.

Sendo assim, é declarada uma variável de nome lista1 que instancia e inicializa o método array( ), parametrizando o mesmo com o tipo de dado dos elementos, seguido da lista de elementos fornecida no enunciado da questão.

Haja visto que a proposta do exercício é exibir em tela alguns dados referentes a nossa array, podemos fazer tal processo diretamente dentro da função print( ).

Sendo assim, como parâmetro para a primeira função print( ) repassamos uma f'string referente ao tamanho da lista, no que diz respeito a seu número de elementos, e esse processo pode facilmente ser realizado fazendo uso da função len( ).

Já para o segundo print( ), na máscara de substituição da f'string instanciamos lista1 aplicando sobre a mesma o método itemsize, do módulo array, que nos retornará o tamanho em bytes do objeto em questão.

Por fim, no terceiro print( ) usamos de outro método do módulo array, chamado buffer_info( ), que quando aplicado a uma variável /

objeto, lendo a partir da posição 0 de seu índice, retorna seu endereço de memória.

Nesse caso, executando o código acima o retorno será:

lista1 é composta de 11 elementos
lista1 possui um tamanho de 4 bytes
lista1 está alocada na memória no endereço: 3043135766176

---

108 – Escreva um programa que, usando puramente de lógica, encontra todos os números que são divisíveis por 7 e múltiplos de 5, dentro do intervalo de 0 a 500 (incluindo tais valores).

```
numeros = []

for numero in range(0, 501):
    if (numero % 7 == 0) and (numero % 5 == 0):
        numeros.append(str(numero))

print(', '.join(numeros))
```

De acordo com a proposta da questão, usando puramente de lógica, podemos abstrair tal problema simplesmente usando de uma estrutura condicional composta que valida os valores dentro do critério estabelecido e do intervalo definido para a questão.

Sendo assim, inicialmente criamos uma variável de nome numeros que inicialmente recebe como atributo uma lista vazia a ser atualizada recebendo os números validados.

Em seguida criamos um laço de repetição, usando do método range( ) para percorrer o intervalo proposto, retornando a cada ciclo um valor para a variável temporária numero. Lembrando que como aqui os números 0 e 500 precisam serem considerados, devemos parametrizar range( ) com 0 e 501 para que o laço for realmente percorrado número 0 até o número 500.

Dentro do laço de repetição, criamos uma estrutura condicional composta, onde a primeira proposição lógica diz respeito a se o resto da divisão do valor de numero por 7 é igual a 0, do mesmo modo, a segunda proposição se refere a se o resto da divisão do valor de numero por 5 for igual a zero. Lembrando que entre as proposições estamos inserindo o marcador and, definindo assim que para que o valor de numero seja de fato validado, o retorno das duas proposições deve ser True.

Na sequência simplesmente instanciamos a variável numeros, atualizando o conteúdo da mesma inserindo os números validados através do método append( ).

Por fim, criamos uma sentença interna a nossa função print( ) para que o conteúdo da variável numeros seja exibido em tela separando cada elemento com uma vírgula e espaço.

Executando esse código o retorno será:

0, 35, 70, 105, 140, 175, 210, 245, 280, 315, 350, 385, 420, 455, 490

---

109 – Escreva um programa que gera um número aleatório entre 0 a 10, salvando esse número em uma variável, onde o usuário é convidado a interagir tentando adivinhar qual foi o número gerado até acertar. Caso o usuário acerte o número, exiba uma mensagem o parabenizando, também exibindo incorporado em uma mensagem o número em si.

```
import random

num_gerado, num_adivinhado = random.randint(0, 11), 0

while num_adivinhado != num_gerado:
    num_adivinhado = int(input('Digite um número entre 0 e 10: '))

print('Parabéns, você adivinhou o número!!!')

print(f'Número gerado: {num_gerado}')
```

A solução deste problema pode ser alcançada de diversas formas, principalmente da parte referente a criar uma aplicação que executa repetidamente até que o critério definido tenha sido alcançado, visando uma forma reduzida, vamos tratar da parte lógica do problema por meio de um laço de repetição While True.

Para o processo de geração de números aleatórios vamos fazer uso de uma das funcionalidades da biblioteca random. Sendo assim, importamos a biblioteca random uma vez que a mesma não vem carregada por padrão.

Em seguida criamos duas variáveis, num_gerado e num_adivinhado, respectivamente, onde para num_gerado atribuímos a função random.randint( ) por sua vez parametrizada com 0 e 11, em justaposição, para que de fato sejam gerados números inteiros dentro do intervalo de 0 a 10.

A variável num_adivinhado inicialmente recebe como valor 0, valor este a ser atualizado posteriormente.

Dando sequência criamos o laço de repetição, onde em forma de estrutura condicional definimos que enquanto o valor de

num_adivinhado for diferente do valor de num_gerado, é solicitado ao usuário que o mesmo digite um número via função input( ).

Pela lógica, quando o valor de num_gerado for igual ao valor de num_adivinhado, a estrutura de repetição é validada como True, saindo do ciclo de repetição.

Por fim, cumprindo o enunciado da questão, são exibidas via print( ) duas mensagens ao usuário, uma o parabenizando, seguido de outra revelando qual teria sido o número gerado para aquela ocasião.

Executando o código o retorno será:

```
Digite um número entre 0 e 10: 1
Digite um número entre 0 e 10: 3
Digite um número entre 0 e 10: 4
Digite um número entre 0 e 10: 5
Digite um número entre 0 e 10: 6
Digite um número entre 0 e 10: 7
Parabéns, você adivinhou o número!!!
Número gerado: 7
```

---

110 – Escreva um programa que executa um bloco de código situado dentro de um comentário.

```
codigo = '''
num = int(input('Digite um número: '))
def eleva_ao_quadrado(num):
    return num ** 2
print(f'{num} elevado ao quadrado é: {eleva_ao_quadrado(num)}')
'''

exec(codigo)
```

Quando estamos aprendendo os conceitos básicos da linguagem Python, aprendemos que um comentário é um bloco de código ignorado pelo interpretador da linguagem, porém, nada impede que com as ferramentas certas possamos executar algo situado dentro de um comentário, desde que o mesmo esteja atribuído a uma variável.

Em nosso exemplo, inicialmente é criada uma variável de nome codigo, que recebe como atributo um comentário onde dentro do mesmo temos uma variável de nome num, que via input( ) pede ao usuário que digite um número.

Ainda dentro do comentário temos uma função de nome eleva_ao_quadrado( ), que recebe como parâmetro o valor da variável num, retornando tal valor elevado ao quadrado como o próprio nome da função sugere.

Por fim, como última estrutura situada dentro do comentário temos uma função print( ), parametrizada com uma f'string para que dentro de suas máscaras de substituição seja incorporado a mensagem o valor de num assim como o valor de num elevado ao quadrado.

De volta ao escopo global do código, agora temos a parte referente ao mecanismo que executará as instruções situadas dentro do comentário atribuído a variável código.

O processo é bastante simples, bastando chamar a função embutida exec( ), parametrizando a mesma com a variável código.

O método exec( ) por sua vez tem como uma de suas características o poder de executar códigos independentemente de qual origem.

Ao rodar esse código em nossa IDE o retorno será:

Digite um número: 8
8 elevado ao quadrado é: 64

---

111 – Escreva um programa que recebe uma palavra qualquer do usuário, aplicando sobre a mesma uma formatação de estilo, retornando a mesma com marcadores para negrito, itálico e sublinhado. As funções que aplicam os estilos devem ser separadas entre si, porém executadas em conjunto via decoradores para uma função principal.

```
def negrito(palavra):
    def transforma():
        return "<b>" + palavra() + "</b>"
    return transforma

def italico(palavra):
    def transforma():
        return "<i>" + palavra() + "</i>"
    return transforma

def sublinhado(palavra):
    def transforma():
        return "<u>" + palavra() + "</u>"
    return transforma

@negrito
@italico
@sublinhado
def aplica_estilo():
    palavra = input('Digite uma palavra: ')
    return palavra

print(aplica_estilo())
```

De acordo com o problema proposto neste exercício, teremos que criar as devidas funções para aplicar formatação, tornando as mesmas decoradas e executadas a partir de uma função principal. Em função disso podemos presumir que o foco desta aplicação é, além da formatação em si de uma string, realizar tal feito da forma mais eficiente possível.

Decoradores são estruturas de código bastante abstratas, porém, uma vez que dominamos seu entendimento de fato podemos tornar o uso de funções em conjunto algo bastante eficiente, além de mais

prático do que estruturar funções aninhadas para o mesmo propósito.

Sendo assim, partindo para a resolução, logo de início criamos uma função de nome negrito( ) que receberá obrigatoriamente uma palavra a ser convertida.

Dentro do corpo dessa função criamos uma nova função de nome transforma( ), que simplesmente retorna a aplicação de formatação para netrito. A própria função transforma é retornada para que a mesma possa ser aplicada por meio de um decorador.

Exatamente o mesmo processo, nos mesmos moldes, é feito para as funções que aplicam as formatações referentes a itálico e a sublinhado.

Dando sequência, nas linhas 16, 17 e 18 de nosso código criamos os devidos marcadores para as funções negrito( ), itálico( ) e sublinhado( ), seguido da função aplica_estilo( ), que atuará como a função principal que instancia e inicializa as funções decoradas.

A função aplica_estilo( ) por sua vez, dentro de seu corpo instancia uma variável local, mesmo objeto a ser repassado como parâmetro para as funções decoradas, de nome palavra, que via input( ) pede ao usuário que digite uma palavra, retornando essa mesma palavra, que implicitamente será retornada após ter sido usada pelas funções decoradas.

Por fim, diretamente como parâmetro para função prnt( ) repassamos a função aplica_estilo( ) sem a necessidade de nenhum parâmetro adicional.

Executando nosso código o retorno será:

Digite uma palavra: Python
<u>Python</u>

---

112 – Escreva um programa que lê uma palavra ou frase digitada pelo usuário, retornando o número de letras maiúsculas e minúsculas da mesma. Usar apenas de lógica e de funções embutidas ao sistema.

```
texto = input('Digite um texto: ')

maiusculas = 0
minusculas = 0

for letra in texto:
    if (letra .islower()):
        minusculas += 1
    elif (letra.isupper()):
        maiusculas += 1

print(f'No texto {texto} foram encontradas {maiusculas} letras maiúsculas')

print(f'No texto {texto} foram encontradas {minusculas} letras minúsculas')
```

Aqui temos mais um exercício de fácil resolução, uma vez que das ferramentas disponíveis pelo núcleo da linguagem já temos todo o necessário para realizar as devidas verificações de letras maiúsculas e minúsculas.

De início é criada uma variável de nome texto, que como atributo receberá algo digitado pelo usuário através da função input( ).

Em seguida são criadas duas variáveis de nomes maiusculas e minusculas, respectivamente, com valor inicial zero pois serão variáveis de controle a serem atualizadas posteriormente.

Na sequência devemos percorrer cada caractere do texto digitado pelo usuário para assim identificar e diferenciar as letras maiúsculas e minúsculas do mesmo. Para isso, usamos de um laço for que percorre o conteúdo da variável texto, retornando a cada ciclo de repetição um dos elementos atribuídos a variável temporária letra.

Dentro do laço de repetição podemos implementar o mecanismo de verificação, usando de uma simples estrutura condicional que verifica se o último dado/valor atribuído a variável temporária letra é

minúscula aplicando sobre a mesma o método islower( ). Caso tal condição for verdadeira, o conteúdo da variável minusculas é incrementado em uma unidade.

Caso a primeira condição não seja validada como verdadeira, uma segunda condição é imposta, dessa vez verificando se o dado/valor atribuído a letra é maiúsculo aplicando sobre tal variável o método isupper( ). Caso seja, a variável maiusculas é incrementada em uma unidade.

Por fim, são criadas duas funções print( ) onde fazendo uso de f'strings incorporamos a mensagem ao usuário o texto digitado pelo usuário e o número de caracteres classificados como maiúsculos e minúsculos.

Executando nosso código, o retorno será:

```
Digite um texto: Programação em Python
No texto Programação em Python foram encontradas 2 letras maiúsculas
No texto Programação em Python foram encontradas 17 letras minúsculas
```

---

113 – Escreva um programa que verifica se uma determinada palavra consta em um texto de origem. Texto: “Python é uma excelente linguagem de programação!!!”.

```
texto = 'Python é uma excelente linguagem de programação!!!'

pesquisa = input('Digite uma palavra a ser pesquisada: ')

if (texto.find(pesquisa) == -1):
    print(f'Palavra "{pesquisa}" não encontrada no texto de origem.')
else:
    print(f'Palavra "{pesquisa}" é uma das palavras do texto de origem')
```

Existem diversas formas de se buscar uma determinada “palavra” dentro de um texto, haja visto que todo e qualquer conjunto de caracteres alfanuméricos delimitados por aspas em Python são automaticamente reconhecidos como strings. Dessas formas, temos desde funções embutidas do núcleo da linguagem até expressões regulares.

O ponto é que, de acordo com a complexidade do problema proposto, podemos optar pelo método o qual julgarmos mais eficiente para tal finalidade. No caso desse exercício, simplesmente buscar uma determinada palavra em um texto, não há necessidade de realizar tal feito de modo complexo, pois usando das ferramentas básicas da linguagem podemos chegar ao resultado que precisamos de forma simples e objetiva.

Logo de início é criada uma variável de nome texto, que recebe em formato de string o texto proposto no enunciado do exercício.

Em seguida criamos uma variável de nome pesquisa, que através de input( ) receberá do usuário uma palavra específica a ser verificada no conteúdo de texto.

Dando sequência criamos uma estrutura condicional onde, aplicando o método find( ) em nossa variável texto, por sua vez parametrizado com o conteúdo de palavra, verificando se o retorno do método for igual a -1 (lembrando que para esse caso o retorno 0

seria o equivalente a True, onde a palavra em questão foi encontrada no texto, enquanto o retorno -1, equivalente a False, indicando que a palavra em questão não foi encontrada).

Para essa condição, através de nossa função print( ) informamos ao usuário que a palavra em questão não foi encontrada no texto, incluindo a palavra em uma máscara de substituição da f'string.

Caso a condição anterior não seja validada, é exibido ao usuário uma mensagem informando o mesmo que a palavra informada consta no texto de origem.

Nesse caso, o retorno será:

Digite uma palavra a ser pesquisada: programação
Palavra "programação" é uma das palavras do texto de origem

Simulando o erro:

Digite uma palavra a ser pesquisada: JavaScript
Palavra "JavaScript" não encontrada no texto de origem.

---

114 – Escreva um programa que verifica se um endereço de site foi digitado corretamente, validando se em tal endereço constam o prefixo ‘www.’ E o sufixo ‘.com.br’, pedindo que o usuário digite repetidas vezes o endereço até o mesmo for validado como correto.

```
endereco = ''

while ((endereco.startswith('www.') and endereco.endswith('.com.br'))) != True:
    endereco = input('Digite o endereço do site: ')

print(f'{endereco} é um endereço válido.')
```

Quando estamos tratando de validações, Python nos oferece muitos meios e métodos que facilitam nossa vida, descartando a necessidade de estar criando funções que manualmente realizem as devidas verificações.

No que diz respeito a validar conjuntos de caracteres situados como prefixos e/ou sufixos, métodos como startswith( ) e endswith( ) são muito úteis graças a sua facilidade de implementação.

Sendo assim, partindo para a solução, logo de início declaramos uma variável de nome endereço, que inicialmente recebe como atributo uma string vazia, essa variável servirá como variável de controle dentro de nosso mecanismo de validação.

Na sequência, uma vez que o enunciado da questão pede que o endereço deve ser digitado até que seja validado como correto, podemos realizar o processo usando de uma estrutura de repetição While True.

Nessa estrutura criamos uma estrutura condicional composta para validar individualmente se a string fornecida como endereço se inicia com ‘www.’ assim como se a mesma termina com ‘.com.br’. Nesse caso, a primeira condicional aplica sobre a variável endereco o método startswith( ) por sua vez parametrizado com a string ‘www.’, também criamos a segunda condicional onde aplicado a variável endereco está o método endswith( ), parametrizado com ‘.com.br’.

Entre as duas sentenças inserimos o operador lógico and, para que ambas as condicionais sejam levadas em consideração para que, caso seu retorno seja diferente de True, a variável endereco é instanciada, dessa vez usando da função input( ) para pedir que o usuário digite novamente o endereço. Dessa forma, o ciclo se repetirá até que o endereço digitado pelo usuário seja validado como True pelas duas condicionais.

Por fim, usando da função print( ) exibimos em tela via f'strings uma mensagem ao usuário referente a quando o endereço foi considerado como correto.

Executando o código, o resultado será:

Digite o endereço do site: google.com.br
Digite o endereço do site: www.google
Digite o endereço do site: www.google.com.br
www.google.com.br é um endereço válido.

---

115 – Escreva um programa que recebe uma lista de elementos mistos, trocando de posições apenas o primeiro e o último elemento. Lista de referência [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X'].

```
lista1 = [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X']

print(f'Lista original: {lista1}')

temp = lista1[0]
lista1[0] = lista1[-1]
lista1[-1] = temp

print(f'Lista final: {lista1}')
```

Usando puramente de lógica, haja visto que não há necessidade de se usar de alguma função de alguma biblioteca para tal fim, vamos buscar inverter as posições do primeiro e do último elemento da lista conforme é pedido no exercício.

Inicialmente é criada uma variável nomeada como lista1, que recebe como atributo a lista repassada no enunciado da questão.

Em seguida, apenas para fins de comparação, através da função print( ) exibimos a configuração inicial da lista atribuída a lista1.

Logo após, criamos uma variável de nome temp, que recebe como atributo o dado / valor situado na posição 0 de índice de lista1.

Na sequência instanciamos lista1, mais especificamente seu elemento de posição de índice 0, atualizando seu valor com o valor do último elemento de lista1 através do marcador -1 como posição de índice.

Por fim, lista1 em sua última posição de índice recebe como novo valor o valor que havíamos guardado anteriormente em temp.

Exibindo em tela novamente o conteúdo de lista1 via print( ) nos é retornado:

Lista original: [11, 22, 33, 44, 55, 66, 77, 88, 99, 'X']
Lista final: ['X', 22, 33, 44, 55, 66, 77, 88, 99, 11]

116 – Escreva um programa que realiza a validação de uma senha digitada pelo usuário. O mecanismo de validação deve verificar se a senha digitada possui um tamanho mínimo de 6 caracteres, tamanho máximo de 16 caracteres, ao menos uma letra entre a e z, ao menos uma letra maiúscula e ao menos um símbolo especial.

```
import re

senha = input('Insira sua nova senha: ')
x = True

while x:
    if (len(senha) < 6 or len (senha) > 16):
        print('A senha deve conter entre 6 e 16 caracteres')
        break
    elif not re.search("[a-z]", senha):
        print('A senha deve ter ao menos uma letra minúscula')
        break
    elif not re.search("[0-9]", senha):
        print('A senha deve conter ao menos um número')
        break
    elif not re.search("[A-Z]", senha):
        print('A senha deve conter ao menos uma letra maiúscula')
        break
    elif not re.search("[$#@*!&]", senha):
        print('A senha deve conter ao menos um caractere especial')
        break
    elif re.search("\s", senha):
        break
    else:
        print('Senha cadastrada com sucesso!!!')
        x = False
        break

if x:
    print('Senha inválida!!!')
```

Quando estamos falando de mecanismos de validação de senhas, uma série de critérios podem ser definidos, implicitamente exigindo

que o usuário crie ou digite uma senha dentro de uma certa complexidade, por motivos óbvios de segurança.

Para validar cada um dos critérios podemos facilmente criar estruturas condicionais para cada um dos mesmos, enquanto para o processo de ler uma string em busca dos padrões de caracteres exigidos podemos implementar nas próprias condicionais expressões regulares simples.

Sendo assim, logo de início realizamos a importação da biblioteca re, haja visto que a mesma não vem carregada por padrão.

Em seguida declaramos uma variável de nome senha, que através da função input( ) recebe do usuário uma senha a ser digitada pelo mesmo que ficará atribuída a variável senha.

Também criamos uma variável de nome x, que recebe como atributo o valor booleano True, pois esta será uma variável de controle que utilizaremos a seguir como gatilho para encerramento das estruturas condicionais.

Partindo para as estruturas condicionais em si, visando facilitar o processo, podemos criar estruturas condicionais dentro de uma estrutura de repetição while, para que seu bloco de código seja validado somente quando todas as condições impostas forem validadas como True.

Indentado para While, inicialmente criamos uma estrutura condicional composta que valida se o tamanho de senha é menor que 6 ou maior que 16, exibindo em tela uma mensagem orientando o usuário sobre este critério. Apenas para fins de exemplo, é inserido o marcador break para que o código encerre sua execução neste ponto, porém, em uma aplicação real seria interessante instanciar a variável senha, exigindo que o usuário digitasse a senha até que a condição imposta fosse cumprida.

Na sequência é criada uma nova estrutura condicional, dessa vez usando do método re.search( ) (expressão regular), verificando se no conteúdo atribuído à variável senha existem caracteres de a a z,

exibindo em tela via print( ) uma mensagem caso este critério não seja cumprido.

Nos mesmos moldes são criadas outras estruturas condicionais que, fazendo uso de expressões regulares, verificam se na senha digitada existem números de 0 a 9, letras maiúsculas e símbolos especiais.

Quando todos os critérios de validação forem cumpridos, é exibida uma mensagem de senha cadastrada com sucesso, a variável de controle x é instanciada e tem seu dado / valor atualizado para False, encerrando o processo via break.

Já fora da estrutura ehile, é criada uma última estrutura condicional, que verifica o último dado / valor atribuído à variável x. Lembrando que pela leitura léxica o código é lido sequencialmente, logo, esta última verificação do status de x sempre ocorrerá após os mecanismos de validação serem devidamente aplicados.

Executando esse código o retorno será:

Insira sua nova senha: &eG340o0!
Senha cadastrada com sucesso!!!

Simulando o erro:

Insira sua nova senha: 12345
A senha deve conter entre 6 e 16 caracteres
Senha inválida!!!

Insira sua nova senha: 123456
A senha deve ter ao menos uma letra minúscula
Senha inválida!!!

Insira sua nova senha: 1a2b3c4d
A senha deve conter ao menos uma letra maiúscula
Senha inválida!!!

Insira sua nova senha: 1A2b3C4d
A senha deve conter ao menos um caractere especial
Senha inválida!!!

---

117 – Escreva um programa que realiza a contagem do número de segundas feiras que caem no dia 1º de cada mês, dentro do intervalo do ano 2010 até o ano de 2020.

```
from datetime import datetime

segundas = 0
meses = range(1, 13)

for ano in range(2010, 2021):
    for mes in meses:
        if datetime(ano, mes, 1).weekday() == 0:
            segundas +=1

print(f'Entre 2010 e 2020 existem {segundas} segundas-feiras no primeiro dia do mês.')
```

Graças ao módulo datetime, praticamente todas as operações possíveis de serem realizadas em dados em formato de data e hora são possíveis através de métodos pré-moldados de fácil aplicação.

No caso deste exercício, descobrir o número de segundas-feiras, que coincidem com o dia 1º do mês, dentro de um intervalo de tempo definido, parece ser algo complexo de ser abstraído, porém, o caminho é usar do método weekday( ) do módulo datetime associado ao método embutido range( ).

Sendo assim, como de costume, logo na primeira linha de nosso código importamos do módulo datetime a ferramenta datetime.

Em seguida é criada uma variável de nome segunda, com valor inicial atribuído 0, pois a mesma será uma variável de controle, tendo seu valor atualizado posteriormente à medida que "encontramos" o número de segundas feiras de acordo com o enunciado da questão.

Também criamos uma variável de nome meses, que recebe como atributo o intervalo de 12 meses de um ano através do método range( ) por sua vez parametrizado em justaposição com 1 e 13.

Logo após, damos início ao desenvolvimento do mecanismo que encontrará os devidos dados que precisamos. Para isso, criaremos

alguns laços for que percorrerão não somente o intervalo de tempo mas o número de meses dentro desse intervalo, buscando apenas as segundas-feiras.

O primeiro laço for percorre via range( ) o intervalo entre o ano de 2010 até o ano de 2020, retornando a cada ciclo de repetição um dado / valor para a variável temporária ano.

Dentro do laço principal criamos um laço de repetição secundário que percorre o intervalo definido anteriormente para a variável meses, retornando a cada ciclo um dado / valor para a variável temporária mes.

Indentado dentro desses laços de repetição aninhados, criamos uma estrutura condicional que verifica se o retorno do método datetime( ), por sua vez parametrizado em justaposição com os dados de ano, mês e 1 (referente a especificamente o primeiro dia do mês) validado como segunda-feira via método weekday( ) igual a 0. Caso essa condição seja verdadeira, a variável segundas tem seu valor incrementado em uma unidade.

Dessa forma, percorreremos todos os meses de todos os anos verificando se no dia primeiro de cada mês é de fato uma segunda feira, contabilizando as mesmas.

Por fim é exibido uma mensagem ao usuário por meio da função print( ), onde usando de f'strings incorporamos na mensagem o número de segundas-feiras encontrado.

Nesse caso, ao executar o código o retorno será:

Entre 2010 e 2020 existem 18 segundas-feiras no primeiro dia do mês.

---

118 – Escreva um programa que exibe uma mensagem em formato de contagem regressiva, contando de 10 até 0, com o intervalo de 2 segundos entre uma mensagem e outra.

```
import time

final = 0

total = 10

print('Contagem iniciada...')

while final <= 10:
    print(total)
    time.sleep(2)
    total -= 1
    final = total
    if total == -1:
        print('Contagem encerrada!!!')
        break
```

Para solucionar este exercício, logo de início importamos a biblioteca time, da qual faremos uso do método sleep( ) para criar o intervalo de espera entre uma mensagem e outra, todo o resto do procedimento pode facilmente ser realizado através de um laço de repetição.

Logo em seguida criamos as variáveis de controle final e total, com seus respectivos valores atribuídos 0 e 10, respectivamente.

Apenas por convenção, através da função print( ) criamos uma simples mensagem para que o início da contagem regressiva seja representada ao usuário.

Em seguida criamos uma estrutura de repetição onde enquanto o valor da variável final for igual ou menos que 10, é exibido em tela via print( ) o próprio valor da variável total, seguido do método time.sleep( ) por sua vez parametrizado com 2, para que o sistema realize uma pequena pausa de 2 segundos até dar continuidade ao processamento deste bloco de código.

Na sequência instanciamos a variável total, atualizando seu valor, decrementando seu valor atual em uma unidade.

Também instanciamos a variável final, atualizando seu valor com o último valor atribuído a variável total.

Por fim criamos uma simples estrutura condicional que terá função de validar quando o valor da variável total for igual a -1, que o processo seja encerrado. Caso contrário, uma vez que o valor de final sempre será menor que 10, caso não inserirmos este tipo de gatilho, o laço de repetição tende a entrar em um loop infinito.

Dessa forma, executando o código o retorno será:

```
Contagem iniciada...
10
9
8
7
6
5
4
3
2
1
0
Contagem encerrada!!!
```

---

119 – Escreva um programa que retorna a data e hora da última vez em que um determinado arquivo foi modificado.

```
import os
import time

def ultima_modificacao(arquivo):
    status = os.stat(arquivo)
    data = time.localtime((status.st_mtime))
    ano = data[0]
    mes = data[1]
    dia = data[2]
    hora = data[3]
    minuto = data[4]
    segundo = data[5]

    str_ano = str(ano)[0:]

    if (mes <= 9):
        str_mes = '0' + str(mes)
    else:
        str_mes = str(mes)

    if (dia <= 9):
        str_dia = '0' + str(dia)
    else:
        str_dia = str(dia)

    return (str_dia + '-' + str_mes + '-' + str_ano + ' / ' + str(hora) + ':' + str(minuto) +
':' + str(segundo))

print('Ultima modificação realizada em: ', ultima_modificacao('testes.txt'))
```

Aqui temos um problema que não necessariamente possui uma dificuldade elevada, mas envolve a combinação de vários processos para que se chegue no resultado esperado. E, como bem sabemos, se uma das etapas envolvidas não estiver de acordo, nossa aplicação não desempenhará a função esperada.

Sendo assim, logo de início importamos as bibliotecas os e time, das quais utilizaremos ferramentas para obtenção de status de um arquivo bem como dados referentes a data e hora do mesmo.

Na sequência criamos uma função de nome ultima_modofocacao( ) que receberá obrigatoriamente um arquivo como parâmetro.

Dentro do corpo dessa função vamos criar uma série de variáveis para dados específicos a serem obtidos do arquivo lido, sendo a primeira dessas variáveis uma nomeada como status, que como atributo recebe o método os.stat( ), por sua vez parametrizado com o próprio arquivo. Essa função por sua vez possui a propriedade de acessar e ler os atributos internos de um arquivo gerados pelo sistema operacional.

Também criamos uma outra variável, dessa vez de nome data, que através do método time.localtime( ), parametrizado com status.st_mtime, receberá dos dados da variável status apenas os dados referentes a última data e hora de modificação atribuída para o arquivo.

Uma vez que temos em nossa variável data todos os dados que precisamos, próximo passo é extrair de data cada um dos elementos para suas respectivas variáveis, de modo que teremos cada elemento referente à data e hora em um objeto instanciável e manipulável.

Para isso, criamos uma variável de nome ano, que como atributo recebe o dado / valor de data em sua posição de índice 0, o mesmo é feito para variável mes, que recebe o conteúdo de data na posição 1 de índice, mais uma vez realizado para a variável dia, que recebe como atributo o elemento de posição de índice 2 da variável data.

Da mesma forma, são criadas as variáveis hora, minuto e segundo, recebendo como atributo os dados / valores dos elementos de índice 3, 4 e 5, respectivamente.

Em seguida vamos criar algumas validações simples de formatação dos dados extraídos, para que possamos retornar os mesmos nessa função em um formato padronizado.

Na linha 14 de nosso código criamos uma variável de nome str_ano, que recebe como atributo em forma de string o conteúdo da variável ano, a partir de sua posição 0 de índice.

Na sequência criamos uma simples estrutura condicional que verifica se o valor de mês é igual ou menos a 9, caso seja, o mesmo é convertido para string e atribuído a variável str_mes, adicionando um 0 na frente do número para que assim tenhamos a padronização de número de mês exibido contendo dois caracteres.

O mesmo é feito validando o formato lido para a variável dia, retornando o valor de dia atribuído para a variável str_dia.

Por fim, criamos o retorno da função, que nesse caso retorna os conteúdos de str_dia, str_mes e str_ano, seguido de hora, minuto e segundo convertidos para o formato de string.

Encerrando a resolução do exercício, é criada uma função print( ), onde dentro da mesma chamamos a função ultima_modificacao( ), parametrizada com o arquivo em questão o qual queremos verificar.

Nesse caso, o retorno será:

Ultima modificação realizada em: 30-10-2021 / 16:14:38

---

120 – Escreva um programa que extrai os dados / valores dos elementos de uma tupla, que representa uma escala de dias de atendimento, para suas respectivas variáveis individuais. Exiba em tela uma mensagem referente aos dias 3 e 4 da escala.

```
dias_atendimento = 2, 3, 4, 5, 8, 9, 10, 11

print(f'Dias de atendimento: {dias_atendimento}')
#print(type(dias_atendimento))

d1, d2, d3, d4, d5, d6, d7, d8 = dias_atendimento

print(f'Não estaremos atendendo nos dias {d3} e {d4}')
```

Quando estamos trabalhando com dados em formato de tupla, sabemos que tais dados são "imutáveis" porém nada impede que realizemos a extração dos mesmos para variáveis as quais podemos iterar ou manipular livremente.

Para fins de exemplo, criamos uma variável de nome dias_atendimento, que recebe como atributo uma série de números que de acordo com a notação serão automaticamente identificados como elementos de uma tupla.

Através da função print( ), usando de uma f'string exibimos uma mensagem referente aos dias de atendimento, exibindo tais dias de acordo com dias_atendimento.

Em seguida criamos 8 variáveis de nomes d1, d2, d3, d4, d5, d6, d7 e d8, que recebem como atributo o conteúdo da variável dias_atendimento. Nesse processo, o que ocorrerá é que os elementos de dias_atendimento serão desempacotados, cada um para sua respectiva variável.

Por fim, de acordo com o enunciado da questão, exibimos uma mensagem, novamente via print( ), com uma mensagem onde na mesma estão incorporados os dados / valores das variáveis d3 e d4, respectivamente.

Executando o código, o retorno será:

Dias de atendimento: (2, 3, 4, 5, 8, 9, 10, 11)
Não estaremos atendendo nos dias 4 e 5

121– Crie uma função reduzida para separar elementos negativos e positivos de uma lista, sem aplicar qualquer tipo de ordenação sobre os mesmos.

```
lista1 = [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2]

print(f'Lista original: {lista1}')

lista2 = [x for x in lista1 if x < 0] + [x for x in lista1 if x >= 0]

print(f'Lista rearranjada: {lista2}')
```

De acordo com o enunciado da questão, devemos simplesmente separar números situados em uma lista entre números negativos de números positivos, usando de uma função reduzida, sem ordenar os números em si de acordo com algum critério. Logo, podemos usar de duas funções lambda que percorrem os elementos de lista1, inicialmente separando os negativos, aplicando o mesmo processo em uma segunda função lambda para separar os positivos, concatenando os resultados dessas funções.

Dessa forma, logo de início temos uma variável de nome lista1 que possui como atributo uma lista com uma série de elementos aleatórios, sendo que em tais elementos temos valores positivos e negativos.

Em seguida é criada uma nova variável de nome lista2, como atributo para a mesma inicialmente criamos uma estrutura reduzida de laço for, que percorre todos os elementos menores que zero, retornando os mesmos para a variável temporária x a cada ciclo de repetição. O mesmo processo é feito para uma segunda expressão lambda onde por meio de um laço for extraímos cada elementos de lista 1 que for igual ou maior que 0. Por fim, concatenamos os resultados como atributo de lista2.

Por fim, exibindo em tela o conteúdo de lista2 via função print( ), exibimos a lista com os mesmos elementos mas agora devidamente separados.

Executando esse código o retorno será:

Lista original: [-1, 6, -9, -8, 4, 0, -3, 2, 7, 1, 8, -2]
Lista rearranjada: [-1, -9, -8, -3, -2, 6, 4, 0, 2, 7, 1, 8]

---

122 – Escreva um programa que retorna os 5 elementos mais comuns de um texto (caracteres individuais), exibindo em tela tanto os elementos em si quanto o número de incidências de tais elementos.

```
from collections import Counter

texto = 'A Radiologia é a ciência que estuda a anatomia por meio de radiações'

a1, a2, a3, a4, a5, a6 = Counter(texto).most_common(6)

print(f'O elemento mais comum é: "{a2[0]}", repetido {a2[1]} vezes')
print(f'O elemento mais comum é: "{a3[0]}", repetido {a3[1]} vezes')
print(f'O elemento mais comum é: "{a4[0]}", repetido {a4[1]} vezes')
print(f'O elemento mais comum é: "{a5[0]}", repetido {a5[1]} vezes')
print(f'O elemento mais comum é: "{a6[0]}", repetido {a6[1]} vezes')
```

Para contabilizar elementos de um contêiner de dados, podemos automatizar o processo por meio do módulo Counter da biblioteca collections, que possui ferramentas dedicadas a esta finalidade.

Então, logo de início importamos da biblioteca collections Counter.

Na sequência temos o texto em si atribuído a uma variável de nome texto.

Como queremos extrair e contabilizar os 5 elementos que mais se repetem, criamos 6 variáveis, presumindo que o elemento que mais se repete é o espaço e este não queremos contabilizar, seguido de outras 5 variáveis onde atribuiremos os elementos válidos contabilizados.

Dessa forma são criadas as variáveis a1, a2, a3, a4, a5 e a6, que recebem como atributo o dado / valor retornado do método Counter( ), parametrizado com a variável texto, filtrando apenas os elementos mais comuns através do método most_common( ).

Por fim, criamos algumas funções print( ), cada uma exibindo em uma f'string um texto, assim como o elemento e o número de repetições do mesmo através da notação de posição de índice, haja

visto que Counter( ) retorna um conjunto contendo o elemento em si na posição 0 e o número de repetições na posição 1.

Executando esse código, o retorno será:

O elemento mais comum é: "a", repetido 11 vezes
O elemento mais comum é: "i", repetido 7 vezes
O elemento mais comum é: "o", repetido 5 vezes
O elemento mais comum é: "e", repetido 5 vezes
O elemento mais comum é: "d", repetido 4 vezes

---

123 – Escreva um programa que recebe três conjuntos numéricos de mesmos valores, o terceiro conjunto deve ser uma cópia literal do segundo. Exiba em tela os conteúdos dos conjuntos, seu identificador de memória e se alguns destes conjuntos compartilham do mesmo identificador (se internamente são o mesmo objeto alocado em memória).

```
import collections

c1 = (2, 4, 6, 8, 10)
dq1 = collections.deque(c1)

c2 = (2, 4, 6, 8, 10)
dq2 = collections.deque(c2)

dq3 = dq2

print(f'Conjunto 1: {dq1}')
print(f'Endereço de Conjunto 1: {id(dq1)}')

print(f'Conjunto 2: {dq2}')
print(f'Endereço de Conjunto 2: {id(dq2)}')

print(f'Conjunto 3: {dq3}')
print(f'Endereço de Conjunto 3: {id(dq3)}')

if (dq1 == dq2) and (id(dq1) == id(dq2)):
    print('Conjunto 1 e Conjunto 2 possuem os mesmos elementos e são o mesmo objeto alocado em memória')
if (dq1 == dq3) and (id(dq1) == id(dq3)):
    print('Conjunto 1 e Conjunto 3 possuem os mesmos elementos e são o mesmo objeto alocado em memória')
if (dq2 == dq3) and (id(dq2) == id(dq3)):
    print('Conjunto 2 e Conjunto 3 possuem os mesmos elementos e são o mesmo objeto alocado em memória')

if (dq1 == dq2) and (id(dq1) != id(dq2)):
    print('Conjunto 1 e Conjunto 2 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
if (dq1 == dq3) and (id(dq1) != id(dq3)):
    print('Conjunto 1 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
```

```
if (dq2 == dq3) and (id(dq2) != id(dq3)):
    print('Conjunto 2 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória')
```

Para esse exercício temos um sistema de validação interessante. Pois teremos que equivaler elementos internos a tuplas, assim como seus identificadores de objeto alocado em memória.

Como sempre, se tratando de Python temos diversas formas de solucionar um determinado problema, cabendo ao desenvolvedor escolher a forma a qual julga mais performática para sua aplicação.

Para realizar as devidas comparações literais, vamos optar por transformar nossos conjuntos de elementos em objetos deque.

De início declaramos uma variável de nome c1, que recebe uma tupla com 5 elementos numéricos sem nenhuma particularidade.

Também criamos uma variável de nome dq1 que via collections.deque( ) converte a tupla atribuída para c1 em deque.

Em seguida criamos outra variável de nome c2 com sua respectiva tupla que, de acordo com o enunciado pela questão, deve ter os mesmos elementos do primeiro conjunto.

Da mesma forma como feito anteriormente, convertemos c2 para deque atribuindo este dado para a variável dq2.

Encerrando essa etapa, criamos uma variável de nome dq3 que recebe como atributo uma cópia de dq2.

Em seguida são criadas algumas funções print( ) para exibir tanto os conteúdos dos conjuntos quanto seus identificadores de memória via id( ).

Para finalizar a solução de nosso problema, podemos criar algumas estruturas condicionais compostas realizando as devidas equivalências entre os dados propostos.

Sendo assim, logo na linha 20 de nosso código, criamos a primeira estrutura condicional, verificando se o conteúdo de dq1 é igual ao conteúdo de dq2, assim como se o identificador de dq1 é o mesmo

identificador de dq2, onde caso essas duas proposições forem verdadeiras, é exibido em tela via print( ) uma mensagem ao usuário dizendo que tanto o conteúdo quanto o identificador de dq1 e de dq2 são iguais.

O processo é repetido criando novas estruturas condicionais, verificando as equivalências de dados e de identificadores entre as variáveis dq1, dq2 e dq3.

Não havendo nenhum erro de sintaxe, executando nosso código o retorno será:

Conjunto 1: deque([2, 4, 6, 8, 10])
Endereço de Conjunto 1: 3015981294592
Conjunto 2: deque([2, 4, 6, 8, 10])
Endereço de Conjunto 2: 3015975677760
Conjunto 3: deque([2, 4, 6, 8, 10])
Endereço de Conjunto 3: 3015975677760
Conjunto 2 e Conjunto 3 possuem os mesmos elementos e são o mesmo objeto alocado em memória
Conjunto 1 e Conjunto 2 possuem os mesmos elementos, porém são objetos diferentes alocados em memória
Conjunto 1 e Conjunto 3 possuem os mesmos elementos, porém são objetos diferentes alocados em memória

---

124 – Crie uma ferramenta implementável em qualquer código que realiza a contagem de número de variáveis locais presentes em uma determinada função do código.

```
def minha_funcao():
    x = 1
    y = 2
    nome = 'Tania'
    lista = []
    print(f'{x}, {y}')
    print(f'{nome}')
    print(f'{lista}')

print(minha_funcao())

var_locais = minha_funcao.__code__.co_nlocals

print(f'A função {minha_funcao} possui {var_locais} variáveis')
```

Mais uma vez, aqui temos um problema de simples resolução, desde se conheçam as ferramentas certas. Uma maneira de se inspecionar o conteúdo interno estrutural de um objeto em Python, nesse caso, uma função, contabilizando suas variáveis internas, é fazendo o uso do método interno co_locals.

Para o exercício em si, é criada uma função de nome minha_funcao( ) com algumas variáveis e seus respectivos dados / valores atribuídos.

Na sequência, apenas por convenção criamos uma função print( ) que parametrizada com minha_funcao( ), executa a função no momento em que a função print( ) é inicializada.

Em seguida, é criada uma variável de nome var_locais, que recebe como atributo a função minha_função, declarada sem os parênteses para que assim a mesma não seja executada, seguido de __code__ e do método interno co_nlocals. Dessa forma, é realizada uma simples inspeção sobre a função minha_funcao( ), lendo suas estruturas internas e contando o número de variáveis locais.

Por fim, através de uma função print( ), usando de f'strings, exibimos uma mensagem ao usuário, incorporando na mensagem o marcador da função em si, assim como o número de variáveis locais conforme pedido no exercício.

Executando o código, o retorno será:

1, 2
Tania
[]
None
A função <function minha_funcao at 0x000001C2CA12D160> possui 4 variáveis

---

125 – Escreva um programa que recebe do usuário um nome e um telefone, verificando se o nome digitado consta em uma base de dados pré-existente, caso não conste, cadastre esse novo nome e telefona na base de dados.

```
clientes = {'Fernando' : '991357259',
            'Alberto' : '981120491'}

novo_cliente = input('Digite o nome do cliente: ')

if novo_cliente in clientes.keys():
    print(f'{novo_cliente} já existe na base de dados')
    novo_cliente = input('Digite o nome do cliente: ')
else:
    print(f'{novo_cliente} não está cadastrado.')
    print('Digite novamente o nome a ser cadastrado: ')
    nome = input('Digite o nome: ')
    telefone = str(input('Digite o telefone: '))
    clientes.__setitem__(nome, telefone)

print(clientes)
```

Quando estamos trabalhando com dicionários, sempre devemos nos ater a sua sintaxe e métodos compatíveis, para que não seja gerado nenhuma exceção a partir do mesmo.

Por parte de sintaxe devemos sempre lembrar que ao inserir dados manualmente em um dicionário, já devemos os inserir seguindo a notação de chave : valor.

Diretamente ao código, inicialmente criamos uma variável de nome clientes que recebe como atributo um dicionário inicialmente composto de dois itens.

Logo em seguida criamos uma nova variável de nome novo_cliente que como atributo recebe algo digitado pelo usuário via input( ).

Em seguida criamos uma estrutura condicional onde, se o dado / valor atribuído a variável novo_cliente estiver presente nas chaves de clientes, é exibido em tela uma mensagem ao usuário via print( ), chamando novamente a função input( ) para que o mesmo tenha a oportunidade de digitar outro nome.

Caso a condição imposta acima não for válida, então é exibido via print( ) uma mensagem de orientação ao usuário, seguido de duas variáveis de nomes nome e telefone, respectivamente, que usando de input( ) pedem que o usuário digite tais dados.

Por fim, atualizando o conteúdo de clientes, aplicamos sobre tal dicionário o método __setitem__( ), por sua vez parametrizado em justaposição com nome e telefone, que alimentarão os campos chave e valor, respectivamente.

Por fim, é exibido o conteúdo de clientes via print( ) para que assim possamos ver o dicionário atualizado com os novos dados.

Rodando nosso código o retorno será:

Digite o nome do cliente: Alberto
Alberto já existe na base de dados
Digite o nome do cliente: Maria
Maria não está cadastrado.
Digite novamente o nome a ser cadastrado:
Digite o nome: Maria
Digite o telefone: 991258813
{'Fernando': '991357259', 'Alberto': '981120491', 'Maria': '991258813'}

---

126 – Crie uma calculadora de 7 operações (soma, subtração, multiplicação, divisão, exponenciação, divisão inteira e módulo de uma divisão) com toda sua estrutura orientada à objetos.

```
class Calculadora():
    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2

    def soma(self):
        return self.num1 + self.num2
    def subtracao(self):
        return self.num1 - self.num2
    def multiplicacao(self):
        return self.num1 * self.num2
    def divisao(self):
        return self.num1 / self.num2
    def exponenciacao(self):
        return self.num1 ** self.num2
    def divisao_inteira(self):
        return self.num1 // self.num2
    def modulo(self):
        return self.num1 % self.num2

num1 = int(input('Digite o primeiro número: '))
num2 = int(input('Digite o segundo número: '))
operacao = Calculadora(num1, num2)

print('Escolha 1 para SOMA')
print('Escolha 2 para SUBTRAÇÃO')
print('Escolha 3 para MULTIPLICAÇÃO')
print('Escolha 4 para DIVISÃO')
print('Escolha 5 para EXPONENCIAÇÃO')
print('Escolha 6 para DIVISÃO INTEIRA')
print('Escolha 7 para MÓDULO')

escolha = int(input('Digite o número da operação: '))
if escolha == 1:
    print(f'O resultado da soma entre {num1} e {num2} é: ', operacao.soma())
elif escolha == 2:
    print(f'O resultado da subtração entre {num1} e {num2} é: ',
operacao.subtracao())
elif escolha == 3:
```

```
    print(f'O resultado da multiplicacao entre {num1} e {num2} é: ',
operacao.multiplicacao())
elif escolha == 4:
    print(f'O resultado da divisão entre {num1} e {num2} é: ', operacao.divisao())
elif escolha == 5:
    print(f'O resultado da exponenciação de {num1} elevado a {num2}ª potência é: ',
operacao.exponenciacao())
elif escolha == 6:
    print(f'O resultado da divisão inteira entre {num1} e {num2} é: ',
operacao.divisao_inteira())
elif escolha == 7:
    print(f'O resultado do módulo da divisão entre {num1} e {num2} é: ',
operacao.modulo())
else:
    print('Opção inválida')
```

Um dos exercícios clássicos de quando estamos aprendendo programação, independentemente da linguagem, é criar calculadoras para por intermédio delas aprender sobre as interações entre variáveis, operadores, etc...

Para aplicações reais pode ser necessário criar tais estruturas diretamente em formato orientado à objetos para assim tornar o código reduzido e mais performático, uma vez que cada operação matemática pode ser um método de classe interagindo sobre objetos instanciados pela própria classe.

A nível de código criar a calculadora e suas funcionalidades é bastante fácil, permitindo diversas formas de se criar cada um dos operadores, novamente cabendo ao desenvolvedor utilizar do método o qual julgar melhor para tal propósito.

Iniciando a escrita do código, de início já criamos uma classe de nome Calculadora, e dentro do corpo da mesma seu método construtor __init__( ) que nesse caso receberá a própria instância da classe, criando os objetos / atributos de classe num1 e num2 que serão comuns para todas as operações matemáticas.

A seguir é criado o primeiro método de classe personalizado, de nome soma( ), que recebe objetos da instância da classe,

retornando o resultado da soma entre num1 e num2.

Exatamente da mesma forma são criados os métodos de classe subtracao( ), multiplicacao( ), divisao( ), exponenciacao( ), divisao_inteira( ) e modulo( ), alterando o retorno de cada método de classe de acordo com a operação matemática a ser realizada, pela notação de seu referente operador.

Em seguida são criadas as variáveis num1 e num2, que através de input( ) recebem valores digitados pelo usuário, validados como dados de tipo inteiro apenas por convenção. (é perfeitamente normal usar de números de casas decimais, caso seja o propósito, a única particularidade a observar é que não realizando tal validação pode ser que os valores digitados pelo usuário sejam reconhecidos por padrão como do tipo string, não permitindo assim a realização das operações matemáticas).

Também é criada uma variável de nome operacao, que instancia a classe Calculadora, repassando como argumentos para a mesma o conteúdo de num1 e de num2 em justaposição.

Por fim, apenas para criar uma interação com o usuário um pouco mais elaborada, criamos um simples menu através de funções print( ) orientando o usuário a escolher uma das operações através de seu número.

Na sequência criamos uma cadeia de estruturas condicionais, onde de acordo com o número escolhido pelo usuário, a devida operação matemática é selecionada, e diretamente como parâmetro da função print( ), fazendo uso de f'strings, inserimos em suas respectivas máscaras de substituição os valores de num1, num2 e a operação matemática em si, que no caso, exibirá neste campo apenas o retorno gerado pela função matemática escolhida.

Não havendo nenhum erro de sintaxe, ao executar o código o retorno será:

Digite o primeiro número: 7
Digite o segundo número: 3
Escolha 1 para SOMA

Escolha 2 para SUBTRAÇÃO
Escolha 3 para MULTIPLICAÇÃO
Escolha 4 para DIVISÃO
Escolha 5 para EXPONENCIAÇÃO
Escolha 6 para DIVISÃO INTEIRA
Escolha 7 para MÓDULO
Digite o número da operação: 5
O resultado da exponenciação de 7 elevado a 3ª potência é: 343

---//---

Digite o primeiro número: 100
Digite o segundo número: 7
Escolha 1 para SOMA
Escolha 2 para SUBTRAÇÃO
Escolha 3 para MULTIPLICAÇÃO
Escolha 4 para DIVISÃO
Escolha 5 para EXPONENCIAÇÃO
Escolha 6 para DIVISÃO INTEIRA
Escolha 7 para MÓDULO
Digite o número da operação: 4
O resultado da divisão entre 100 e 7 é: 14.285714285714286

---

127 – Escreva um programa que recebe uma fila composta de valores referente a idades de pessoas, extraindo os 3 elementos de maior idade para uma outra fila prioritária, sem que os elementos prioritários sejam rearranjados.

```
import heapq as hq

fila_original = [23, 45, 60, 19, 23, 50, 18, 82, 46, 77]

fila_normal = hq.nsmallest(7, fila_original)

fila_prioritaria = list(set(fila_original) - set(fila_normal))

print(f'Fila Normal: {fila_normal}')

print(f'Fila Prioritária: {fila_prioritaria}')
```

Sempre que estamos a trabalhar com filas, é interessante que realizemos a manipulação dos elementos das mesmas através de ferramentas específicas que não somente automatizam o processo como também gerenciam regras para comportamentos específicos a serem aplicadas a seus elementos.

Visando solucionar o problema dessa questão, vamos tentar realizar o processo de separação dos elementos de maior valor para uma outra lista da forma mais reduzida o possível, e para isso vamos fazer uso de heapq.

Partindo para o código, inicialmente importamos o módulo heapq o referenciando como hq apenas por convenção.

Logo em seguida temos uma variável de nome fila_original, que atribuído para si possui uma lista com 10 elementos representando idades de pessoas conforme o enunciado da questão pede.

Para realizarmos a separação dos elementos que não entrarão na fila prioritária, criamos uma variável de nome fila_normal, que recebe como atributo o retorno da função hq.nsmallest( ), por sua vez parametrizada com 7, haja visto que dos 10 elementos da fila original filtraremos 3 para a fila prioritária, seguido da variável que é a fila de origem.

Uma vez que temos uma variável com todos os elementos e uma com os elementos não prioritários, podemos simplesmente realizar a subtração entre suas respectivas variáveis para assim separar e atribuir a uma nova variável os elementos prioritários.

Sendo assim, criamos uma nova variável de nome fila_prioritaria que recebe em forma de lista a subtração dos elementos de fila_original e fila_normal. Lembrando que para realizar tal subtração dos elementos internos das listas, sem alterar suas posições, inicialmente convertemos as listas para sets, depois aplicamos a subtração.

Por fim, através de uma função print( ) exibimos o conteúdo de fila normal.

Da mesma forma, através de print( ) exibimos o conteúdo de fila_prioritaria, e assim concluímos a resolução dessa questão.

Executando o código o retorno será:

Fila Normal: [18, 19, 23, 23, 45, 46, 50]
Fila Prioritária: [82, 60, 77]

CONCLUSÃO

Como costumo escrever no fechamento de meus livros... E assim concluímos esse pequeno tratado, dessa vez de exercícios resolvidos e comentados em Python.

Espero que esses exercícios e seus respectivos comentários tenham lhe ajudado a entender os conceitos mais comumente usados quando estamos programando em Python.

Como comentei logo no início deste livro, programar envolve muita prática, e meu grande compromisso com você aqui foi na medida do possível tentar desmistificar muitos dos conceitos que podem gerar dificuldades no estudante de programação.

Caso você já possuísse alguma bagagem de conhecimento sobre os tópicos abordados, espero que tais exemplos tenham ao menos lhe ajudado a ter uma ótica diferente sobre os problemas propostos, de modo que agregasse a seu conhecimento.

Por fim, resta meu sincero agradecimento por ter adquirido esta humilde obra e que a leitura da mesma tenha sido tão prazerosa a você quanto foi a mim por escrevê-la.

Um forte abraço.

Fernando Feltrin

LIVROS


Python do ZERO
à Programação Orientada a Objetos
Fernando Feltrin
FERNANDO FELTRIN
PROGRAMAÇÃO ORIENTADA À OBJETOS COM PYTHON
EBOOK
Ciência de Dados e Aprendizado de Máquina
Fernando Feltrin
ARRAYS COM PYTHON + NUMPY
FERNANDO FELTRIN
FERNANDO FELTRIN
INTELIGÊNCIA ARTIFICIAL COM PYTHON
FERNANDO FELTRIN
REDES NEURAIS ARTIFICIAIS
FERNANDO FELTRIN
ANÁLISE FINANCEIRA COM PYTHON
FERNANDO FELTRIN
TÓPICOS AVANÇADOS EM PYTHON
FERNANDO FELTRIN
PYTHON
PYTHON + TENSORFLOW
PYTHON NA PRÁTICA
APRENDA PYTHON ATRAVÉS DE EXERCÍCIOS
TÓPICOS ESPECIAIS EM PYTHON
FERNANDO FELTRIN
BLOCKCHAIN E CRIPTOMOEDAS EM PYTHON
FERNANDO FELTRIN
TÓPICOS ESPECIAIS EM PYTHON
VOL. 2
FERNANDO FELTRIN
COLETÂNEA
TÓPICOS ESPECIAIS EM PYTHON
VOL. 1 E 2
2 livros, 280+ páginas
FERNANDO FELTRIN

Python do ZERO à Programação Orientada a Objetos

Programação Orientada a Objetos com Python

Ciência de Dados e Aprendizado de Máquina

Inteligência Artificial com Python

Redes Neurais Artificiais com Python

Análise Financeira com Python

Arrays com Python + Numpy

Tópicos Avançados em Python

Visão Computacional em Python

Python na Prática (Exercícios resolvidos e comentados)

Tópicos Especiais em Python vol. 1

Tópicos Especiais em Python vol. 2

Blockchain e Criptomoedas em Python

Coletânea Tópicos Especiais em Python

Python na Prática (Códigos comentados)

PYTHON TOTAL (Coletânea de 12 livros)

CURSO

[Curso Python do ZERO à Programação Orientada a Objetos]

Mais de 15 horas de videoaulas que lhe ensinarão programação em linguagem Python de forma simples, prática e objetiva.

Curso com selo de classificação mais alta em sua categoria.